TRS-80® MODEL I/III

SERIES I EDITOR ASSEMBLER

Catalog Number

26-2011
26-2013

Radio Shack TRS-80 SOFTWARE™

CUSTOM MANUFACTURED IN THE USA FOR RADIO SHACK

A DIVISION OF TANDY CORP.

TRS-80 Series-I Tape/Disk Editor/Assembler © 1981 Tandy Corporation. All rights reserved.

Derived from original Tape Editor/Assembler © 1978 Microsoft. Licensed to Tandy Corporation.

Series I Editor Assembler Manual © 1981 Tandy Corporation. All rights reserved.

Reproduction or use without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information obtained herein.

Please refer to the Software license on the back of this manual for limitations on use and reproduction of this Software package.

Table of Contents

1. Introduction 1
What is an Editor/Assembler? 1
The Series-I Editor/Assembler 1
The Scope and Organization of This Manual 3
Notation Conventions 4

2. Loading the Editor/Assembler 5
Tape Systems—Level II and Model III BASIC 5
Tape Systems—Level I 5
Disk Systems 6

3. Using the Editor 7

4. Using the Assembler 21

5. Sample Programming Session 31

6. The Z-80 Instruction Set 45

7. Appendices
A. Using the TPSRC Utility (Disk Systems Only) 227
B. ROM and TRSDOS I/O Subroutines 228
C. Z-80 Status Indicators (Flags) 231
D. Numerical Listing of Z-80 Instruction Set 234
E. Alphabetical Listing of Z-80 Instruction Set 240
F. Z-80 CPU Register and Architecture 246
Index 253

Part One:

Introduction

What Is an Editor/Assembler?

An editor/assembler is a two-part program that lets you communicate with a computer in its low-level, "native" language, rather than in some high level, "foreign" language like BASIC or FORTRAN. We call this native language "machine-language."

Using the editor, you enter the machine-language source code, consisting of a convenient set of abbreviations and symbols. The assembler then converts or assembles this into object code, which the Computer understands.

But I thought my TRS-80 spoke BASIC!

Well, you're right, it does. But only because it contains a built-in BASIC interpreter. This interpreter converts or interprets your BASIC programs into object code, which the computer can understand.

With a Built-In Interpreter, Who Needs Machine-Language?

Well, if you—

- Enjoy learning how things—especially, computers—work;
- Want to do things faster than BASIC will allow;
- Want to make the most efficient use of your Computer's memory;
- Want to modify the way your computer inputs and outputs data

—then you need machine-language. (Of course, there are plenty of other reasons you may want to use it.)

The Series-I Editor/Assembler

There are two versions of this software package, one for tape and one for disk systems.

Tape Version

Three cassette tapes are included. One contains EDTASM, which is the Editor/Assembler. Level II and Model III BASIC customers may load and run this tape using BASIC'S SYSTEM command. The second tape contains SYSTEM. This program is for Level I customers with a minimum of 16K memory. It is loaded

with the CLOAD command, and prepares the Level I Computer to load the EDTASM tape. The third tape contains a sample program for tape systems with at least 32K of RAM. If you have only 16K, you can still type in and use the sample program given in Section 5.

Disk Version

Two diskettes are included. There is one in Model I TRSDOS format and one in Model III.

The disk version software includes three programs:

- EDTASM, the Editor/Assembler program
- SAMPLE/SRC, a source listing of all the Z-80 instructions
- TPSRC, a utility to read source tapes written by the tape version of the Editor/Assembler and two write object "SYSTEM" tapes.

The Series-I Editor/Assembler is especially good for beginners of machine language programming. Its commands and features are fairly simple, and it does not require that you understand advanced programming concepts. On the other hand, experienced programmers will find this editor/assembler a workable tool for all but the most complex, large-scale applications.

Features

Editor Features

- Automatic line numbering for convenient source-code entry.
- Line renumbering command with automatic renumbering if necessary.
- Single-letter commands plus optional parameters.
- Global search capability for changing your source text.
- Source text may be saved on tape or disk, depending on your computer system.
- Source files on tape or disk may be loaded or "chained" in memory.
- Source text may be listed to the printer.

Assembler Features

- Controlled by a single-letter command with optional switches.
- Options include: wait on error, no symbol table, list to printer, and trial assembly with no object code output.
- Supports labels up to six characters long.
- Eight pseudo-ops.
- Resides in memory with the Editor, so you can easily go back and forth between editing and assembling.

Scope and Organization of This Book

In this manual, we will show you how to use the Editor/Assembler. Along the way, we'll cover a few principles of assembly-language programming. We'll include a sample program. Even if you don't understand assembly-language programming, you should be able to try out this sample program.

In the next section (*Section 2*), we'll tell you how to load the Editor/Assembler. We'll assume you already know how to start-up your Computer, and to get it to the BASIC READY level (cassette systems) or to the TRSDOS READY level (disk systems). There are separate loading instructions for:

- Tape systems — Level I
- Tape systems — Level II and Model III BASIC
- Disk systems — Models I and III TRSDOS

In *Section 3*, we'll show you how to use the editor. This section is organized for ease of use the first time through. For quick reference later on, there's an alphabetical summary of all editor features at the end of Section 3.

In *Section 4*, we describe the assembler. Here we'll simply explain the assembly command format and syntax. You'll need this information when you get around to writing your own assembly-language programs.

In *Section 5*, we present a sample assembly-language program. We go through all the procedures, from entering the program to loading and executing the assembled version.

Section 6 is a complete Z-80 instruction set — the native language of your TRS-80.

This manual is written for use with Model I or III systems using either tape or disk storage. There are a few operational differences, depending on which system you have. In these cases, we have written separate instructions for the differing systems. Follow those pertaining to your Computer.

What else do I need?

To write your own assembly-language programs, you'll need more information than is contained in this manual. If you know Z-80 or another assembly language, this manual will probably be sufficient. But if you've never done any assembly-language programming, you'll need to do some further study.

Radio Shack sells an ideal book for future TRS-80 assembly-language programmers: TRS-80 Assembly Language Programming, by William Barden, Jr. Its catalog number is 62-2006. Although it was written specifically for the Model I TRS-80, most of it applies as well to the Model III.

Notation and Special Terms Used in This Book

Notations

`COMPUTER TYPE`	Indicates material that is input to or output from the Computer. Note: All computer prompts in this manual are given in uppercase.
italic type	Represents variable information that you provide in a command. (i.e., file names, line numbers, etc.)
(KEY)	Key which you should press. These will not be visible on the screen.
[optional information]	Square brackets enclose optional parts of a command.

Special Terms

source code (or text)	An assembly-language source program you have loaded from tape or disk or typed.
source file	An assembly-language source program you have saved on tape or disk.
object code	The output from the assembler, i.e., coded Z-80 instructions.
object file	Object code stored on tape or disk so that it may be loaded and executed.

Part Two:

Loading the Editor/Assembler

Tape Systems—Level II and Model III BASIC

The Editor/Assembler is a machine-language program stored on tape at 500 baud. Its file name is EDTASM.

1. Turn on your Computer and press (ENTER) to the prompt for memory size. (In Model III systems, first type L to the CASS? prompt.)
2. Get your recorder ready to play the Editor/Assembler tape.
3. Type SYSTEM (ENTER), then EDTASM (ENTER). The Computer will begin loading from the tape. After a successful load (takes about 2 minutes), the *? prompt will reappear.
4. Type / (ENTER). The Editor/Assembler starts by displaying a heading followed by an asterisk at the beginning of the next line. The asterisk is the prompt, telling you the Editor/Assembler is waiting for a command.

Now skip to *Section 3*.

Tape Systems—Level I BASIC

Before you can load the Editor/Assembler tape, you must get your Computer into a "system" mode. The SYSTEM tape does this.

1. Turn on your Computer. It should be in the READY mode.
2. Get your recorder ready to play the SYSTEM tape.
3. Type CLOAD (ENTER). The Computer will begin loading from the tape. After a successful load (takes about 2 minutes), a "PRESS ENTER WHEN CASSETTE IS READY" will appear on the next display line. Your Computer is now in the system mode.
4. Prepare the recorder to play the EDTASM tape.
5. Press (ENTER). The Computer will begin loading from the tape. After a successful load (takes about 2 minutes), the Editor/Assembler will start by displaying a heading followed by an asterisk at the beginning of the next line. The asterisk is the prompt, telling you the Editor/Assembler is waiting for a command.

6. Volume setting may need to be adjusted for a successful load.

Now skip to *Section 3*.

Disk Systems

The program file name for the Editor/Assembler is EDTASM/CMD.

1. Under TRSDOS READY, type: `EDTASM` (ENTER).
2. The Editor/Assembler will start by displaying a heading, followed by an asterisk on the next line. The * is the prompt, indicating the Editor/Assembler is waiting for a command.

Part Three:

Using the Editor

Assuming you have just started the Editor/Assembler, it is displaying an asterisk on the screen. This is the "prompt." It tells you the Editor/Assembler is waiting for a command.

The Editor consists of commands that allow you to create, edit, save and load your source programs. We'll divide these commands into three groups:

- Text-handling — creating and modifying the source program.
- File input/output — saving the program on disk or tape and loading it from disk or tape.
- Miscellaneous — getting the memory status, exiting from the Editor/Assembler.

Special Terms

Before using the commands, we need to define a few special terms used in this section.

"text" is the information (source program) that you have entered into the Computer. The insert command allows you to begin entering text one line at a time, pressing (ENTER) at the end of each line. The Editor automatically numbers each line.

"text buffer" is the area in memory where your text is stored.

"current line" is the line most recently entered, displayed, or referenced in a command.

"file" is the source text stored on tape or disk.

"file name" is the name given to the file. In tape systems, the file name consists of from one to six letters or numbers. In disk systems, the file name follows the rules of TRSDOS file specifications (for full details, see your TRSDOS reference manual):

filename [/ext] [.password] [:d]

"inc" or "increment" refers to the number which is used to compute successive line numbers for your text. When you start the Editor, the increment equals 10.

"line ref" or "line reference" is the way you specify a single line in your text. A line reference may be any number from 0 to 65529, or any of the following special symbols:

#	First line in the text buffer
.	The current line
*	The last line in the text buffer

"line range" indicates a range of lines in your text file; it is a pair of line references separated by a colon.

line-ref:line-ref

"TOF" and "EOF" — refer to top of file (first line) and end of file (end of file). The Editor will use these abbreviations in certain messages to you.

Sample Commands

These examples are simply to show the use of the special terms and notation. The commands are explained later in detail.

P 100	"Print line 100."
P #:.	"Print text from the first line to the current line.
D .	"Delete the current line."
I *line ref.,inc*	"Start inserting at line, using inc as an increment between lines. ("line ref." and "inc" are variables you replace with appropriate values.)

A Few Words about Spaces

In general, spaces are not significant inside editor commands. You may use them or omit them. Exception: No spaces inside a file name, line reference or in the command (F)-Find.

Special Keys

(ENTER)	To complete a command or a line of text, you must press this key.
(BREAK)	To cancel a command or to stop inserting text, press this key. The line that the (BREAK) is pressed is not saved. Press (BREAK) on the line following the last line.
(▲)	Press this key to see the previous line of text.
(▼)	Press this key to see the next line of text.
(◀)	This key erases the previously typed character.
(▶)	This functions as a tab key. You will use it while inserting text. The tab positions are spaced eight columns apart.
LEFT (SHIFT) (◀)	This erases the line you have been typing.
(@)	This causes a pause in a listing or printout. Press any key to continue.

Editor Commands

We'll cover the commands in a typical sequence in which you might use them. For an alphabetical summary, see the end of this section.

Text Handling Commands

Inserting Your Text

When the asterisk is displayed, you may type in a command—not your source text. To enter source text, you must get into the insertion mode.

First, to get your Computer "in step" with our examples, type D #:* (ENTER). That erases any text that you might already have entered into the text buffer.

Now we'll go into the insertion mode. Type I (ENTER). The Computer will display 00100. All we do is type in text for line 100 and press (ENTER). The Computer will automatically provide the next line number.

```
00100  ;  ANY CHARACTERS FOLLOWING A SEMI-COLON (;) IS A
          COMMENT (ENTER)
00110
```

We may continue like this until we finish entering the text. Remember to press (ENTER) at the end of each line.

```
00110  ;  PRESS -> AT THE START OF THE NEXT LINE (ENTER)
00120            RET    ;A VERY SHORT PROGRAM (ENTER)
00130
```

In line 120, we pressed tab (▶) once at the beginning of the line, and once after RET. Tabs are very important in source programs; they are used instead of spaces to separate the standard fields in an assembly-language program. (We'll explain further in part 4.)

That's all the text we want to type in for now, so press (BREAK). The asterisk will reappear on the next line.

Displaying Your Text

To see the text, use the Print command. For example: P #:* (ENTER). This tells the Computer to display all the lines in the text buffer. To see a single line, specify that line, as in: P *100* (ENTER). Another way to display lines one at a time is with (▲) (previous line) and (▼) (next line).

If you omit a line reference, the Computer will display a screenful of lines, starting at the current line. This is a good way to look at a large text file, one screenful at a time. Simply press P (ENTER) to see the next screenful.

Note: If the total file is to be displayed you may execute T (ENTER) prior to Print command to insure that current line is TOF.

Getting a Hard-Copy of the Source Program

To output to a line printer instead of to the display, substitute "H" (hard copy) for "P". For example, the command H #:* prints out the entire source program. If printer is not ready press (BREAK) to return to command line.

(For instructions on getting hard copy of an assembled program, see *Section 4*.)

Adding Lines between Existing Lines

Suppose you want to add a line between lines 100 and 110. Use the Insert command, but specify a starting line number between 100 and 110:

```
I 105
00105 ;THIS LINE IS ADDED (ENTER)
00115 (BREAK)
*
```

When you pressed (ENTER) for line 105, the Computer used the current increment (10) to generate line 115, which will not be between 100 and 110. To insert more than one line between any two lines, you can specify an increment of 1.

For example,

```
I 105,1 (ENTER)
00106
```

Line 105 is already in use, so the Computer gives you the next number, using an increment of 1:

```
00106 ;WE'LL JUST TYPE IN A FEW LINES (ENTER)
00107 ;NOTICE THAT THE INCREMENT OF 1 IS STILL IN USE (ENTER)
00108 ;WHAT WILL HAPPEN WHEN WE REACH LINE 110? (ENTER)
00109 ;THAT LINE IS ALREADY IN USE . . . (ENTER)
00110 ;. . . BUT EDTASM GIVES YOU THAT NUMBER ANYWAY. (ENTER)
00111 (BREAK)
```

A line "collision" was about to occur when you entered line 110, since that number was already in use. So the Editor automatically renumbered all lines.

To begin inserting lines at the end of the file, use the Bottom command, B (ENTER). This makes the current line the last line.

Changing a Line in Your Text

To make a change within a line of text, use the Edit command. This puts you in a special intra-line edit mode in which several useful functions are available. To begin editing a line, type E followed by the line number (or line symbol "#", "*", ".") and press (ENTER). The Computer will display the line number followed by the cursor (blinking block or underline). This is your "working copy" of the line. Changes you make will not take effect until you exit from the intra-line edit mode.

To exit from the intra-line edit mode, press (ENTER) or E (ENTER) and changes are saved. Press (BREAK) or Q (ENTER) and the line remains in its original form.

Here are the functions available in the intra-line edit mode:

(L)	Lists the line in its current form and starts a new working copy on the next line.
n (SPACEBAR)	(Spacebar) Moves the cursor forward *n* spaces, showing the next *n* characters in the line. If *n* is omitted, 1 is used.
(←)	Moves cursor back one space in the line, but does not erase the character from the working copy.
n (S) *c*	(Search) Positions the cursor at the *n*th occurrence of character *c*, counting from the current cursor position. If *n* is omitted, positions to the first occurrence after the current position.
n (D)	Deletes the next *n* characters. If *n* is omitted, 1 is used.
n (K) *c*	(Kill) Deletes all characters up to the *n*th occurrence of character *c*. If *n* is omitted, deletes up to the first occurrence.
n (C) *c1 . . . cn*	Changes the next *n* characters to characters *c1 . . . cn*.
(A)	(Again) Cancels all changes made and lets you edit the line again.
(I) *newtext*	Insert newtext. Insertion will continue until you press (SHIFT) (↑) or (ENTER). While inserting, the (←) key will erase a character, and the (SPACEBAR) will insert a space. You must exit from this insertion function before you can use any of the other editing functions.
(X)	(Extend) Begin inserting at the end of the line.
(H)	(Hack) Delete remainder of the line and begin inserting at the current position.
(ENTER) or (E)	Exits to the * command level. The changes you made will take effect.
(BREAK) or (Q)	(Quit) Exits to the * command level. The changes you made will be canceled.

The best way to learn to use these edit functions is to experiment with them. For example, type E (ENTER) to start editing the current line. The Computer will display the line number. Press (L) to see the line in its current form and start a new working copy. Now try each of the commands listed above.

Remember: To exit from the intra-line editor at any time, press (ENTER). To stop the insertion function but continue editing, press (SHIFT) (↑).

Replacing a Line

You cannot use the Insert command to replace a line, because the Computer will always renumber the lines in case of a line collision. To replace a line, type R followed by the line reference and press (ENTER).

For example, to replace line 100, type: R 100 (ENTER). The Computer will display 00100. Go ahead and type in the new text for this line. When you press (ENTER), the Computer will act just as it does in the line insertion mode: it will compute a new line number using the current increment and renumbering the lines if necessary to avoid a collision. From this point on, you are inserting, not replacing. Only line 100 is replaced.

Deleting Lines

To delete a range of lines, type D line range. For example,

D 100	Deletes line 100
D .	Deletes the current line
D 100:120	Deletes all lines from 100-120
D #:*	Deletes all lines (first to last)

Finding a String within Your Text

The Find command searches through your text for any one word string you specify, and tells you which lines contain the text.

Suppose you have a large text file in memory, and you want to change each occurrence of "LBL" to "LABEL." The Find command will identify each line that contains "LBL." Simply type: T (ENTER) to position the current line to the beginning of the text, then type FLBL (ENTER). The Computer will search for the string of characters immediately following the F and ending with the carriage return ((ENTER)).

The editor will print the line number of the first occurrence of LBL. That line becomes the current line. You may begin editing it by typing E (ENTER).

To find subsequent occurrences of LBL, simply type F (ENTER). The editor continues searching at the current position and remembers the string being searched.

Remember: (1) Type in the search string immediately after the "F" with no spaces, unless the search string starts with spaces. (2) The Find command begins searching at the current line, so set the current line to TOF first if you want to search through the entire text.

Renumbering Your Text

After inserting lines (and having them automatically renumbered), you may want to renumber them "manually." The Number command does this. Type N *start-line*, increment (ENTER). *Start-line* will be the lowest-numbered line in the renumbered program.

For example, the command: `N 1000,10` (ENTER) renumbers the text 1000, 1010, 1020, etc.

After renumbering, the current line is the last line in the file, and the increment is what you specified in the N command.

If no start line is typed, the renumbering will begin with the current line. If no increment is specified, 10 is used.

Source File Input/Output Commands

In this section, we'll show how to save a source program and then reload it. (For instructions on outputting and loading an object file, see Section 4.)

There are three general groups of editor I/O commands:

- Writing the source program to tape or disk
- Loading the source program from tape or disk
- Printing the source program on the display or on a line printer. We've already described these commands (H and P).

Saving the Source Program

Once you have typed in and edited a source program, you should save it on tape or disk. That way, if you ever need to modify the source program, you won't have to retype it; you can simply load it and make changes.

The tape version of Editor/Assembler always assumes you want tape I/O, and the disk version assumes you want disk I/O. (Disk systems may load source tapes via the TAPESRC utility, described later in the appendix.)

Note to Model III Customers: All tape I/O is done at 500 baud, regardless of the cassette baud rate you selected when you started up the Computer.

Tape Systems

1. Using a blank cassette tape, put your recorder into the record mode.
2. Type `W` *file* (ENTER). Use a file name from one to six characters. You may omit the file name, in which case the tape file will be named NONAME.

   Example:

   `W MOVE` (ENTER)

3. The Editor/Assembler will prompt you to get the cassette recorder ready. Be sure it's in the record mode, then press (ENTER). The Editor/Assembler will write the text onto the tape.
4. After writing the tape, the Editor/Assembler will return to the command mode (asterisk).
5. Make at least one additional tape copy of the program.

6. Remove the tape from the recorder and label it. Be sure to identify it as a source tape.

Disk Systems

1. Type W *file* (ENTER). For file, use a standard TRSDOS file name with an optional password and drive specification. The Editor will automatically add the extension /SRC to the file name. To override this, include a different extension in the file specification.

   You may omit the file name, in which case the file will be called NONAME/SRC.

   Example:

   W MOVE (ENTER)

   writes the source program into the file MOVE/SRC.

2. After writing out the file, the Editor will return to the command mode (asterisk).

Loading a Source Program

Tape Systems

1. Prepare the recorder to play the source tape.
2. Type L *file* (ENTER). For file, substitute the correct file name. If there are several files on the tape, the Editor will search through them until it reaches the one you named. You may omit the file name, in which case the first file on the tape will be loaded.

   Before the Editor starts loading from the tape, it will prompt you to get the cassette recorder ready. Press (ENTER) when ready.

3. After loading the source program, the Editor will return to the command mode (asterisk).

Disk Systems

1. Type L *file* (ENTER). For file, specify the file in standard TRSDOS form. If the specification you give does not include an extension, the Editor will automatically use the extension /SRC.

   You may omit the file specification. The Editor will then attempt to load a file named NONAME/SRC.

(If you already have a source program in the text buffer, the Editor will warn you:

```
TEXT IN BUFFER, CHAIN FILES?
```

If you want to add the disk file onto the end of the current text in memory, type Y (ENTER). This will chain the new file onto the end of the file in memory and automatically renumbers the total file. If you don't want to "chain" the files, but wish to erase the current file and load the new one, type N (ENTER).)

2. After loading the file, the Editor will return to the command mode (asterisk).

Miscellaneous Commands

Determining the Memory Status

To find out the size of the current source program and the amount of free memory, type M (ENTER). The status will be shown in bytes.

Exiting from the Editor/Assembler

The quit command (Q (ENTER)) takes you out of the Editor/Assembler and back to TRSDOS or BASIC (if you are in a level II computer). Before using this command, be sure to save your source program, if desired, because you won't be able to recover it simply by restarting the Editor/Assembler.

Editor Error and Warning Messages

BAD PARAMETER(S)	This indicates that you gave the editor an invalid command. Check the syntax used, and the values of parameters given (they may be out of range).
BUFFER FULL	The area assigned to text storage is full. You may be able to split the source text into two modules.
LINE NUMBER TOO LARGE	During the generation of new line numbers (insertion or line renumbering) a line number greater than 65529 was needed. This is too large. Use a smaller line number increment.
NO SUCH LINE	A reference was made to an unused line number.
NO TEXT IN BUFFER	All commands except load, insert, memory-status, and quit require some text to be in the buffer.
STRING NOT FOUND	You issued a find command and the editor could not locate the string you specified. Be sure you had the current line set properly (find begins searching at the current line number).

Editor/Assembler Alphabetical Summary

Special Keys

(ENTER)	Executes the current command.
(BREAK)	Cancels or interrupts a command.
(←)	Erases the last character typed.
(↑)	Displays the previous text line.
(↓)	Displays the next text line.
(SHIFT) (←)	Erases the entire line. (Use left shift key only)
(→)	Tabs forward eight spaces.
(@)	Pauses execution of a command; press again to continue.
(SHIFT) (↑)	Escapes from the character insertion command in the edit mode. (Use left shift key only)

Symbols and Abbreviations

#	First line in text
*	Last line in text
.	Current line in text
line ref	A single line number or line symbol (#, *, or .).
line range	A pair of line refs separated by a colon (line ref : line ref)
inc	An increment between lines.

Commands

A [*file*] [,*switch*. . .]	Assemble. Switches are: LP (line printer, WE (wait on error), NL (no listing), NS (no symbol table), NO (no object code output).
B	List bottom (last) line of text.
D [*line ref or line range*]	Delete line(s).
E [*line ref*]	Edit line ref.
Subcommands	
(L)	Lists working copy of line
n (SPACEBAR)	Advance *n* spaces.
(←)	Backspace 1 space.
n (S) *c*	Search for *nth* occurrence of *c*.
n (D)	Delete next *n* characters.
n (K) *c*	Kill up to *nth* occurrence of *c*.
n (C) *c1* . . . *cn*	Change next *n* characters to *c1* . . . *cn*.
(A)	Cancel changes and start again.
(I) *newtext*	Insert *newtext*. Press (ENTER) or (SHIFT) (↑) to quit.
(X)	Extend line.
(H)	Hack rest of line and begin inserting.
(ENTER) or (E)	Exits to the command level; changes take effect.
(BREAK) or (Q)	Cancels changes and quits editing.
F [*text string*]	Find the *text string* immediately following the letter "F"; or find the current text string. (No space between (F) and text string).
H [*line range*]	List lines on the printer. If printer not ready use (BREAK) to recover.
I [*line ref*] [,*inc*]	Insert at *line ref* using inc. If no *line ref* has been determined 100 is used.
L [*file*]	Load a source *file*.
M	Display memory status.
N [*line ref*] [,*inc*]	Renumber text.
P [*line range*]	List lines on the display.

Q	Quit Editor/Assembler; return to TRSDOS or BASIC (Level II).
R [*line ref*]	Replace line and continue in the line insertion mode.
T	List top (first) line of text.
W [*file*]	Write a source file.

Part Four:

Using the Assembler

In Section 3, we showed you how to type in, edit, and save a source program. For a source program, we used an arbitrarily chosen text.

Now we are ready to discuss the assembler—the software that converts your source text into object code that can be understood by the TRS-80's Z-80 microprocessor, and writes this object code to a tape or disk file. We'll break this section up into three parts:

A. The Assemble command—syntax, options, file output, error conditions, etc.

B. Assembler language—definitions, syntax, input/output format, etc.

If you're new to assembly language, you don't have to read all this now. You may skip to Section 5, which presents a sample programming session. This will give you hands-on experience with the Editor/Assembler. Then, when you come back to this section, you'll have a better idea of what it's all about . . .

The Assemble Command

You enter the Assemble command at the command level (asterisk). It consists of the abbreviation "A" followed by a space and an optional file name and optional switches. (We call them "switches" because they turn various functions on and off.)

There are various combinations of spaces and commas that will work in the assemble command. For simplicity, we'll stick with one workable set of rules for command syntax.

A [*file*] [,*switch* . . .]

The file name and switch are optional. (If no file name is used, you must still type in a space after the "A.") Every switch used must be preceded by a comma. Spaces before or after the file are acceptable and have no effect.

A source program must be originated in RAM or loaded into RAM before it can be assembled.

For example:

A ZAP,NS,NL,WE (ENTER)

"ZAP" is the file name; "NS", "NL" and "WE" are switches. The commas are required. The meaning of this and the following commands will be explained in the following pages.

A ,NO,WE,NS (ENTER)

No file name is given.

As another example:

A (SPACEBAR) (ENTER)

No file name or switches are specified.

File Name

The file name you specify will be assigned to the tape or disk object file. If you omit a file name, "NONAME" will be used. (For further details, see File Output later in this section.)

Switches

If you don't specify any switches in your assemble command, the Assembler will do the following:

- Print the assembly listing on the screen
- Print error and warning messages in the listing without pausing
- Print a symbol table after the listing is completed
- Output the object code to tape or disk, using the file name you specified (or "NONAME" if you omitted one)

Here are the switches available. You may use as many as you want in any order. Remember to put a comma before each switch used.

LP	(Line printer) Output listing, error messages, and symbol table to the line printer, not to the display.
WE	(Wait on error) Pause after each error message; operator presses (ENTER) to continue.
NL	(No listing) Don't output an assembly listing.
NS	(No symbol table) Don't output a symbol table.
NO	(No output) Don't output any object code.

File Output—Disk Systems

If you do not specify the NO switch, and if no terminal errors occur during the assembly, the Assembler will write the object code to the disk file you specify.

Use a standard TRSDOS file name with an optional password and drive specification. The Assembler will automatically add the extension "/CMD" to the file name. To override this, include a different extension in the file specification.

If you omit a file specification, the Assembler will use "NONAME/CMD" as the object file.

Examples:

```
A  ZAP,NO,WE
```

Waits on errors, does not output object code.

```
A  ZAP,LP
```

Outputs the assembly listing to the printer, outputs object code to ZAP/CMD.

Use of Object Files

Every object file is stored in a special format that allows it to be loaded and executed by TRSDOS. An object file cannot be loaded by the Editor/Assembler. (Since it is no longer in text form, the Editor/Assembler can't do anything with it.)

To load and execute an object file program while you are in the TRSDOS READY mode, type the file name and press (ENTER). If the extension is "/CMD," you don't need to include it in the file name.

To load an object file and return to TRSDOS READY, type LOAD *filename* (ENTER). In this case, you must include the extension even if it is "/CMD." For further details on the use of object files, see Section 5.

Now skip ahead to "Assembler Error Messages."

File Output—Tape Systems

Note to Model III Customers: All tape output is done at 500 baud.

If you do not specify the "NO" switch, and if no terminal errors occur, the Assembler will write the object code to cassette tape, using the file name you specify. The file name may be from one to six characters long. If you omit one, "NONAME" will be used.

Before writing the tape, the Assembler will prompt you to get the cassette ready. Using a blank tape, prepare the recorder to record; when ready, press (ENTER). The Assembler will then write the tape.

Make at least two copies of each object file. Remove the cassette and label it as an "object" tape.

Use of Object Tapes

Object tapes are stored in a special format for loading via the SYSTEM command. (Level I systems must first load the SYSTEM tape; then the object tape.) An object file cannot be loaded by the Editor/Assembler. (Since it is no longer in text form, the Editor/Assembler can't do anything with it.)

To load an object tape while in BASIC, type: SYSTEM (ENTER) then *filename* (ENTER) . After the tape has been loaded, you may press (BREAK) to return to BASIC, or / *address* (ENTER) to begin execution at the specified address. If you type / (ENTER), omitting the address, an address specified on the tape itself will be used. (For details, see the Section 5.)

Assembler Error Messages

Four kinds of errors may occur after you enter an assemble command.

1. *Command errors*. If there is an error in your command, no assembly will be attempted. The Assembler will display the message "BAD PARAMETER(S)"
2. *Terminal errors*. During assembly, an unrecoverable error occurred. The assembly is cancelled.

   The only terminal error is "SYMBOL TABLE OVERFLOW." This occurs when there is not enough memory to handle the symbol tables required for assembly. Use a machine with more memory (if possible), or break the program up into modules and assemble them separately.
3. *Fatal errors*. One of the source lines contained an error. No object code is generated for the offending line, but the assembly continues. Here are the terminal errors:

BAD LABEL	Invalid sequence of characters were used as a label. (See "labels.")
EXPRESSION ERROR	An invalid expression was used as an operand. (See "expressions.")
ILLEGAL ADDRESSING MODE	One of the operands used is illegal with the specified Z-80 instruction.
ILLEGAL OPCODE	Unrecognizable characters were used in the opcode (mnemonic) field.
MISSING INFORMATION	Mnemonic or operands are missing.

4. *Warnings*. A probable error occurred, but the assembler will generate object for the offending line anyway. The code may not be what the programmer intended. Warning messages are:

BRANCH OUT OF RANGE	Relative branch instruction outside of the range −126 to +129 bytes. Instruction is assembled to branch to itself.
FIELD OVERFLOW	An operand (number or expression) is out of range for the specified instruction. The operand is set equal to zero.
MULTIPALLY DEFINED SYMBOL	A label has been used to identify two different places or represent two different values. All but the first definition will be ignored.
MULTIPLE DEFINITION	A duplicate operand is used.
NO END STATEMENT	No end statement was found.
UNDEFINED SYMBOL	The operand field contains a symbol which has not been defined. A value of 0 is used for this symbol.

Assembly Language

In the first part of Section 4, we discussed the use of the assemble command. In this part, we'll discuss Assembly as a programming language.

An assembly program is made up of source statements. Each source statement consists of up to four fields. A "field" is a range of columns on the display. We'll agree to consider column 1 to be the first column of source text. Column 1

is the first column after a space that follows the line number. Source statements are written using the I (insert) command.

Field	Column Range
Label	1-6
Mnemonic	9-15
Operand(s)	17-31
Comment	May begin anywhere but must be preceeded by a semi-colon (;).

Labels are used to identify individual source statements. A label may be from one to six characters. It must start with an alphabetical character. For example:

```
MOVE
LOOP
LOOP1
CLS
T1
```

are all valid labels. Labels must start in column 1.

Mnemonics are the abbreviations used to represent Z-80 operations, for example:

```
LD    Load
DEC   Decrement
RET   Return
```

Mnemonics are also called "operation codes" or "opcodes." Mnemonics must start in column 9.

Operands are the values used by certain assembler statements. An operand may be a Z-80 register or I/O port, or a one- or two-byte value. For example:

```
LD    A,3
```

tells the Z-80 to load into register A the number 3. "A" and "3" are operands. Symbols may be used in place of actual numbers. For example:

```
LD    HL,VIDEO
```

tells the Z-80 to load into register HL the value for VIDEO (defined elsewhere in the program). The first operand must start in column 17.

Comments document the program. They are ignored by the assembler. A comment may begin in any column of a source statement, subject to the following limitations: All comments start with a semi-colon, which tells the assembler to ignore the rest of the line.

When you type in a source program, use a tab (⇥ key) to separate the fields, not spaces. This method is faster and saves memory. Furthermore, the tab settings correspond to the first columns in each field.

Example:

```
00100            ; THIS IS A SAMPLE PROGRAM
00110            ;
00120   ;LABEL   MNEM.   OPERAND(S)   COMMENT
00130            ORG     32700        ;FOR 16K MACHINES
00140   BEGIN    LD      HL,3C00H     ;(HL)=VIDEO RAM)
00150            LD      A,'*'
00160            LD      (HL),A       ;WRITE ASTERISK TO VIDEO
00170            RET                  ;RETURN TO CALLER
00180            END                  ;END OF SOURCE PROGRAM
```

Lines 100-120 are comments. Lines 130-170 consists of assembly-language statements followed in most cases by comments.

There should be one tab character at the end of each field. Spaces (entered via (SPACEBAR) should only be used inside comments and inside character constants.

Assembler Statements

There are three kinds of assembler statements:

1. *Pseudo Operations*. Sometimes called "pseudo ops," these statements are not translated into Z-80 object code. They control various functions of the assembler itself, such as defining labels, reserving memory, and setting the programs origination address. Pseudo ops must begin in column 9.
2. *Commands*. These are also directed at the assembler. The Series I Assembler has two assembler commands, *LIST ON and *LIST OFF (described later). These commands must begin in column 1.
3. *Z-80 Operations*. These consist of a mnemonic (sometimes called an operation code or "opcode") sometimes followed by one, two or no operands. They are translated directly into object code. Some Z-80 instructions translate into one byte of object code; others may translate into two, three, or four bytes. The opcode must begin in column 9. Tabbing one time moves to column 9.

Special Terms and Abbreviations for Operands

nnnn or *nn* — Represents a number. For one-byte numbers, *nn* is used. For two-byte numbers, *nnnn* is used. (Two-byte numbers are assembled into two's complement binary values. First comes the least significant byte (LSB), then the most significant byte (MSB)). A number may be any of these:

Decimal number

Hexadecimal number *nnnn*H or *nn*H. The suffix "H" indicates hexadecimal; if the number starts with A-F, prefix a 0 to it, as in 0F0H.

Octal number: *nnnnn*Q or *nnn*O. The suffix "Q" or "O" indicates octal.

Current address, "$" (The address in the program counter will be used in place of the $).

Character constant: Any character inside single quotes. The constant is converted into its ASCII character code. For example, 'A' is converted into 65.

Any numeric expression (see "Expressions").

Pseudo-Operations

`ORG` *nnnn*

(Originate) This sets the address reference counter. It determines where subsequent Z-80 code and data will reside in memory. If no ORG statement is given in your source program, the address reference counter will be set to 0.

ORG should be used before any Z-80 instructions or data storage pseudo ops. It may be repeated. The programs in this manual are ORGed at decimal 32512 (hexadecimal 7F00). All subsequent ORG's are absolute.

symbol `EQU` *nnnn* or *nn*

(Equate) This assigns the value *nnnn* to the symbol. Each time the symbol is used as an operand in the source program, the assembler will replace it with *nnnn*. The EQU statement may appear anywhere in the program. A particular symbol may be equated only once.

label `DEFL` *nnnn*

(Define *label*) This assigns a temporary value *nnnn* to the specified label. The value may be changed as often as required within the source program.

`END` *nnnn*

This indicates the end of a source program. If there are any following lines in the program, they will be ignored. The address *nnnn* sets the entry point to the program. If omitted, the entry to TRSDOS (disk systems) or BASIC (cassette systems) will be used. For details, see section 5.

[*label*] `DEFB` *nn*

This defines the contents of the current address to be *nn*. This pseudo op allows you to initialize the contents of one-byte storage locations used by the program. *nn* may be a one-byte value or a character string enclosed in single-quotes.

[*label*] `DEFW` *nnnn*

This defines the contents of the current two-byte address to be *nnnn*. This pseudo op allows you to initialize the contents of two-byte storage locations used by the program.

[*label*] `DEFS` *nn*

(Define storage) This reserves *nn* bytes of memory, starting at the current address. (The reference address will be incremented by *nn* before the next

source statement is assembled.) This pseudo op allows you to reserve space for buffers, parameters, etc.

[*label*] `DEFM` *string*

(Define message) This stores the specified string of characters, beginning at the current address.

Assembler Commands

The *LIST command allows you to suppress parts of a source listing. Error messages and the offending source statements will still be listed. These commands are very useful when you are debugging long programs, because the parts of the program already corrected do not need to be listed. You may also want to use them to suppress the listing of long tables of data contained in programs (e.g., DEFM strings).

The asterisk (*) portion of the *LIST ON and *LIST OFF command must be in column one.

```
*LIST OFF
```

Has no effect on the assembly, but turns off the assembly listing.

```
*LIST ON
```

Has no effect on the assembly, but turns the assembly on again (after *LIST OFF).

Using Expressions as Operands

The assembler will accept an expression in place of any numeric operand. Expressions include symbols, numeric or string constants, and combinations of these using the arithmetic and logical operators listed below.

+ and − Addition and subtraction. Example:

```
LD    HL,VID+80H
```

− Negation. Example:

```
LD    HL,VID-1
LD    HL,-1 (0 understood)
```

& Logical AND. Example:

```
LD    A,(HL)&0FH
```

< Shift left or right. This operator shifts a value right or left by a specified number of bits, in this format:

value < *nn*

If *nn* is negative, the *value* is shifted to the right and zeroes fill on the left. If *nn* is positive, the *value* is shifted to the left and zeroes fill on the right. Example:

```
LD    A,VAL<2
```

Shifts the VAL two bits to the left and fills with zeroes on the right.

The Z-80 Instruction Set

Section 6 is a full Z-80 instruction set. The Z-80 registers and flags available for the programmer's use and a description of the Z-80 architecture is in Appendix F.

Part Five:

Sample Programming Session

In this section, we'll take you step by step through the Series I Editor/Assembler. Our goal will be to create a machine-language subroutine that may be called from a BASIC program or the disk operating system of your computer.

The machine-language we'll present is simple but useful. Given a source address, a destination address, and a length-value, it will copy a block of memory into another area of memory. Doing this with normal BASIC statements is slow. Doing this with machine-language is almost instantaneous.

Creating the Source Program

Start the Editor/Assembler as explained in Section 2. Then type I (ENTER) to get into the line insertion mode. Now type in the following program, pressing (ENTER) at the end of each line. (Remember to use TAB to space from the end of one field to the start of the next field.)

```
00100 ; SUBROUTINE COPIES ONE BLOCK OF MEMORY TO ANOTHER AREA
00110 ; ON ENTRY, (SRC) = SOURCE ADDRESS
00120 ;           (DST) = DESTINATION ADDRESS
00130 ;           (LEN) = NUMBER OF BYTES TO MOVE
00140         ORG     32512
00150 MOVE    LD      HL,(SRC)          ; SOURCE ADDR.
00160         LD      DE,(DST)          ; DESTINATION ADDR.
00170         LD      BC,(LEN)          ; LENGTH
00180         LDIR
00190         RET
00200 SRC     DEFW    0
00210 DST     DEFW    0
00220 LEN     DEFW    0
00230         END     MOVE
```

Press (BREAK) to quit inserting. Then type P #:* (ENTER) to see the entire source program. If there are any errors, use the edit mode (E command) to correct the line.

If you have a printer, you may get a hard copy of the text by typing H #:* (ENTER).

Now we are ready to make a copy of the source program. We'll call it "MOVE."

Saving/Loading a Source Program (Tape Systems)

Using a blank cassette tape, get the recorder ready to record. Type W MOVE (ENTER). Press (ENTER) again when you are ready to record. After the tape is recorded, the Editor/Assembler will return in the command mode (asterisk). It's a good idea to make a second tape copy.

Now try reloading the program. Delete the text from memory by typing D #:* (ENTER). Then rewind the recorder, prepare it to play, and type L MOVE (ENTER). Press (ENTER) again when the recorder is ready to play. After the program has been loaded, the Editor will return in the command mode. Now skip to the paragraph titled, Trial Assembly.

Saving/Loading a Source Program (Disk Systems)

Type W MOVE (ENTER). After the file is written, the Editor/Assembler will return in the command mode (asterisk). The file will be called MOVE/SRC.

Now try reloading the source program. Delete the text from memory by typing D #:* (ENTER). Then type L MOVE (ENTER). After the source program has been loaded, the Editor will return to the command mode, listing text and memory contents.

Trial Assembly

Now we are ready to see if the program can be assembled without errors. We'll use the NO (no output) and WE (wait on errors) switches for this purpose.

The source program should be in memory. Type A ,NO,WE (ENTER). The Editor/Assembler will put the assembly listing on the screen. If any errors are found, the listing will be paused. An error message will appear directly above the offending line. Press any key to continue.

If any assembly errors were found, use the edit mode to correct them, and try another trial assembly.

If you have a printer, you may request a hard copy of the assembly listing. This will be preferable to the display listing, since most listings require more than 64 columns per line. To output to the printer, type: A ,NO,LP (ENTER).

Figure 1 shows the assembly listing generated by our sample program. We've added callouts to identify the various fields in the listing.

```
Memory     Object     Line
Loc.       Code       Number   Label    Mnemonic   Operand(s)
                      00100    ; SUBROUTINE COPIES ONE BLOCK OF MEMORY TO ANOTHER AREA
                      00110    ; ON ENTRY, (SRC) = SOURCE ADDRESS
                      00120    ;           (DST) = DESTINATION ADDRESS
                      00130    ;           (LEN) = NUMBER OF BYTES TO MOVE
7F00                  00140                ORG        32512
7F00       2A0E7F     00150    MOVE        LD         HL,(SRC)       ; SOURCE ADDR.
7F03       ED5B107F   00160                LD         DE,(DST)       ; DESTINATION ADDR.
7F07       ED4B127F   00170                LD         BC,(LEN)       ; LENGTH
7F0B       EDB0       00180                LDIR
7F0D       C9         00190                RET
7F0E       0000       00200    SRC         DEFW       0
7F10       0000       00210    DST         DEFW       0
7F12       0000       00220    LEN         DEFW       0
7F00                  00230                END        MOVE
00000      Total Errors

LEN     7F12
DST     7F10
SRC     7F0E
MOVE    7F00
```

Symbol Table

Figure 1. Sample Assembly Listing

Here are a few comments on the source program (line references are to column 3 of the listing):

Line 140 sets the origination address of the program. We've chosen an address near the top of memory in a 16K RAM system. If you change this address, be sure to make the appropriate changes in the BASIC calling program (presented later).

Line 230 ends the program. Since we gave an operand (MOVE), the Editor/Assembler will store the value of MOVE as the entry address to the program. If we had omitted an operand here, the entry address to the program would have been set to address 0000H. (More later.)

Object Code Output

After confirming that the program can be assembled without errors, we are ready to create the object file on tape or disk. We'll use an assemble command that outputs object code only.

Tape Systems

Using a blank tape, prepare the recorder to record. Type `A MOVE,NL,NS` (ENTER). Press (ENTER) again when ready. The Editor/Assembler will write out the object tape. It's a good idea to repeat this process to get a second tape copy.

Disk Systems

Type `A MOVE,NL,NS` (ENTER). The Editor/Assembler will create an object file named MOVE/CMD.

Running the Sample Program

Our sample program, MOVE, may be executed as a BASIC subroutine or as an independent program.

First, we'll try it as a BASIC subroutine.

Tape Systems (Level II and Mod III only — will not execute in a Level I machine)

Start BASIC and answer the MEMORY SIZE question by typing 32511 (ENTER). This will keep BASIC from using the area where the subroutine will reside.

Now load the subroutine:

Type SYSTEM (ENTER). Prepare the recorder to play the object tape, then type MOVE (ENTER). After the program has been loaded, the `*?` will return. Press (BREAK) to return to BASIC. Now type in the BASIC program given in Listing #1. (Page 36)

Run the program. Specify any source address, and specify a destination between 15360 and 16383. Specify any length from 1 to 1024. However, the destination + length must not exceed 16384.

The program will copy a block of memory beginning at the source onto video memory beginning at the destination. The number of bytes copies will be the length value.

Disk Systems

Start TRSDOS. Under TRSDOS READY, load the subroutine by typing `LOAD MOVE/CMD`.

Start BASIC. Answer the MEMORY SIZE question by typing `32511` (ENTER). This will keep BASIC from using the memory where MOVE resides.

Now type in the program given in Listing 2. (Page 36)

Run the program. Specify any source address, and specify a destination between 15360 and 16383. Specify any length from 1 to 1024. However, the destination + length must not exceed 16384.

The program will copy a block of memory beginning at the source onto video memory beginning at the destination. The number of bytes copied will be the length value.

Executing a Machine-Language Program Directly

MOVE is a subroutine called from a BASIC program. However, you can also execute machine-language programs created with the Editor/Assembler.

Disk Systems

Under TRSDOS READY, type in the program name and press (ENTER). The program will be loaded and executed, starting at the address specified in the END statement of the original source listing (e.g., line 230 of our sample program). Don't use our sample program this way; it was designed to be called from BASIC only.

Tape Systems (Level II and Mod III BASIC)

Load the program using the SYSTEM command, as explained previously. After the program has been loaded from tape, the *? will reappear. Don't press (ENTER). Press / (ENTER) instead. The Computer will begin executing the program at the address specified in the END statement of the original source listing (e.g., line 230 of our sample program).

Alternatively, you may type / address (ENTER) to override this entry address.

(Don't try this with MOVE; that subroutine should only be called from a BASIC program like the one we presented.)

Tape Systems (Level I users)

You may load the program using the Level I 'System Loader' tape that came with your EDTASM. This is accomplished by typing CLOAD. A prompt "CASSETTE READY" will appear on the screen. When the tape is ready to load press (ENTER). Your object program will load at this time. The Computer will begin executing your program at the address specified in the END statement.

You may write your own "System Loader" and put it at the beginning of each Level I program. (Refer to Appendix B) Tapes loaded into Level I with the "System Loader" must be ORGed above 4500H and be created by EDTASM.

```
10 POKE 16526,0: POKE 16527,127
20 SRC = 32526
30 DST = 32528
40 LN = 32530
50 CLS
60 INPUT "SOURCE"; S
70 INPUT "DESTINATION"; D
80 INPUT "LEN"; L
90 IF (D<15360) OR (D>16383) THEN 230
100  VL = S: MM = SRC: GOSUB 190
110 IF (D<15360) OR (D>16383) THEN 230
120 IF D+L > 16384 THEN 240
130 VL = D: MM = DST: GOSUB 190
140 VL = L: MM = LN: GOSUB 190
```

```
150 X = USR(0)
160 IF INKEY$="" THEN 160
170 GOTO 50
180 'BREAK NUMBER INTO MSB, LSB
190 MS% = VL/256: LS% = VL - (MS% * 256)
200 'PUT DATA INTO MEMORY
210 POKE MM, LS%: POKE MM+1, MS%
220 RETURN
230 PRINT "INVALID DESTINATION": STOP
240 PRINT "DATA BLOCK EXCEEDS END OF VIDEO RAM": STOP
```

Listing #1.

```
10 DEFUSR = &H7F00
20 SRC = &H7F0E
30 DST = &H7F10
40 LN = &H7F12
50 CLS
60 INPUT "SOURCE"; S
70 INPUT "DESTINATION"; D
80 INPUT "LEN"; L
90 IF (D<15360) OR (D>16383) THEN 230
100 VL = S: MM= SRC: GOSUB 190
110 IF (D<15360) OR (D>16383) THEN 230
120 IF D+L > 16384 THEN 240
130 VL = D: MM = DST: GOSUB 190
140 VL = L: MM = LN: GOSUB 190
150 X = USR(0)
160 IF INKEY$="" THEN 160
170 GOTO 50
180 'BREAK NUMBER INTO MSB, LSB
190 MS% = VL/256: LS% = VL - (MS% * 256)
200 'PUT DATA INTO MEMORY
210 POKE MM, LS%: POKE MM+1, MS%
220 RETURN
230 PRINT "INVALID DESTINATION": STOP
240 PRINT "DATA BLOCK EXCEEDS END OF VIDEO RAM": STOP
```

Listing #2.

Part Six:

The Z-80 Instruction Set

Notation and Other Conventions

This section includes a detailed description of all the Z-80 assembly language instructions. The first line of each of these pages shows the assembly language opcode mnemonic followed by its operand(s). Some instructions have no operands at all. Other instructions have one or two operands. Anything which is capitalized should be copied exactly when you use the editor to write the assembly language source code. Anything shown in lowercase letters will be replaced by an appropriate register, number, or label. For example, the first instruction described in the eight-bit load group is:

LD r,r′

LD is the mnemonic for the Load instruction. If you wish to move the contents of register H into register A, the actual source code is

LD A,H

This should be read as "load register A with the contents of register H."

A detailed explanation of the operand notation is given below, but in general you should note that single lowercase letters are used for eight-bit numbers or registers and double lowercase letters are used for 16-bit numbers or registers. Also note that parentheses around a register pair indicates that the register pair is to be used as a pointer to a memory location. For example, the instruction INC HL means that 1 is to be added to the HL register pair. The instruction INC (HL) means that 1 will be added to a number in memory whose address is found in register pair HL.

Symbol	Specifies one of the registers
r	A, B, C, D, E, H, or L.

Symbol	Specifies a register pair
qq	BC, DE, HL, or AF
ss	BC, DE, HL, or SP
dd	BC, DE, HL, or SP
pp	BC, DE, IX, or SP
rr	BC, DE, IX, or SP

Symbol	Specifies a number or symbol in the range
n	0 to 255 (one byte)
nn	0 to 65535 (two bytes)
d	−128 to 127 (one byte)
e	−126 to 129 (one byte)

Symbol	Specifies any of the following
s	r, n, (HL), (IX + d), or (IY + d)
m	r, (HL) (IX + d), or (IY + d)
(nn)	Specifies the contents of memory location **nn**
b	Specifies an expression in the range (0,7)
cc	Specifies the state of the Flags for conditional JR, JP, CALL and RET instructions

Instruction Format Examples With Explanation

Format Example 1

LD r,(HL)

Operation: r ⇦ (HL)

This is the shorthand description of the instruction. The arrow indicates that data is moved into register r.

When you write the assembly language code, the lowercase r will be replaced by A, B, C, D, E, H or L.

Format:

Mnemonic: LD **Operands:** r,(HL)

Object Code:

0	1	r	r	r	1	1	0

The object code for this instruction is one byte long. To figure out the object code, replace bits 3, 4 and 5 with the appropriate numbers from the table. For example:

Source Code	Object Code
LD A,(HL)	01111110
LD B,(HL)	01000110
LD C,(HL)	01001110

This instruction uses two machine (M) cycles. The first machine cycle consists of four timing (T) states and the second machine cycle consists of three T states for a total of seven T states. In the TRS-80 one T state takes .5636714 microseconds because the clock speed is 1.774038 MHz, for Model I, 4 MHz for Model II and 2.02752 MHz for Model III. The execution time (E.T.), in microseconds, is calculated for the TRS-80. (One microsecond is 10^{-6} seconds or 1/1,000,000 of a second.)

Description:

The eight-bit contents of memory location (HL) are loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If register pair HL contains the number 75A1H, and memory address 75 A1H contains the byte 58H, the execution of

LD C, (HL)

will result in 58H in register C.

Format Example 2

JP cc,nn

Operation: IF cc TRUE, PC ⇦ nn

The jump is made only if the condition cc is true. The arrow indicates that the number nn is moved into the program counter PC. This will cause the program to jump to address nn.

When you write the assembly language code, cc will be replaced by one of the following: NZ, Z, NC, C, PO, PE, P or M. nn will be replaced by a number from 0 to 65535 or a label.

Format:

Mnemonic: JP **Operands:** cc, nn

Object Code:

1	1	cc	cc	cc	0	1	0

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Note: The first n operand in this assembled object code is the low order byte of a two-byte memory address.

The object code for this instruction is three bytes long. To figure out the object code, replace bits 3, 4 and 5 of the first byte with the appropriate number from the table. The second two bytes of the object code are the address being jumped to. For example:

Source Code	Object Code	
JP NZ, 0FF00H	11000010	C2H
	00000000	00H
	11111111	FFH
JP M, 1002H	11111010	FAH
	00000010	02H
	00010000	10H

Note that the low order, or right hand byte, of the address comes first in the object code.

Description:

If condition cc is true, the instruction loads operand nn into register pair PC (Program Counter), and the program continues with the instruction beginning at address nn. If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status bits which correspond to condition bits in the Flag Register (register F). These eight status bits are defined in the table below which also specifies the corresponding cc bit fields in the assembled object code.

The Relevant Flag column shows the value the flag must have if the jump is to occur.

cc	Condition	Relevant Flag
000	NZ non zero	Z = 0
001	Z zero	Z = 1
010	NC no carry	C = 0
011	C carry	C = 1
100	PO parity odd or no overflow	P/V = 0
101	PE parity even or overflow	P/V = 1
110	P sign positive	S = 0
111	M sign negative	S = 1

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the Carry Flag (C flag in the F register) is set and the contents of address 1520 are 03H, after the execution of

JP C,1520H

the Program Counter will contain 1520H, and on the next machine cycle the CPU will fetch from address 1520H the byte 03H. In other words, program execution jumps to the instruction at 1520H.

Format Example 3

CPIR

Operation: $A - (HL), HL \Leftarrow HL + 1, BC \Leftarrow BC - 1$

The shorthand description indicates that three different things are happening:

1. BC is decremented
2. HL is incremented
3. A byte in memory is subtracted from the A register (but the results are not saved).

Format:

Mnemonic: CPIR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	1	B1

The assembly language instruction has no operands.

The object code is two bytes long.

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL is incremented and the Byte Counter (register pair BC) is decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A ≠ (HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero before the execution, the instruction will loop through 64K bytes, if no match is found. Also, interrupts will be recognized after each data comparison.

For BC ≠ 0 and A ≠ (HL):

M cycles: 5 T states: 21(4,4,3,5,5) 4 MHz E.T.: 5.25

For BC = 0 or A = (HL):

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

The total execution time of this instruction depends on how long it takes to find the byte being searched for and the length of the block being searched. If the instruction loops three times before BC = 0 or A = (HL), then there will be 58 (2x21 + 16) timing (T) states executed.

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if A = (HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if BC becomes zero; reset otherwise
N:	Set
C:	Not affected

Example:

If the HL register pair contains 1111H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1111H) : 52H
(1112H) : 00H
(1113H) : F3H

then after the execution of

CPIR

the contents of register pair HL will be 1114H, the contents of the Byte Counter will be 0004H. Since BC ≠ 0, the P/V flag is still set. This means that it did not search through the whole block before the instruction stopped. Since a match was found, the Z flag is set.

The CPIR instruction will affect five of the six condition codes.

Z-80 Instruction Set
Table of Contents

8 Bit Load Group 47
16 Bit Load Group 65
Exchange, Block Transfer
and Search Group 87
8 Bit Arithmetic and Logical Group 105
General Purpose Arithmetic
and CPU Control Groups 135
16 Bit Arithmetic Group 141
Rotate and Shift Group 151
Bit Set, Reset
and Test Group 177
Jump Group 189
Call and Return Group 201
Input and Output Group 211

8 Bit Load Group

LD r,r′

LoaD

Operation: r ⇦ r′

Format:

Mnemonic: LD **Operands:** r, r′

Object Code:

0	1	r	r	r	r′	r′	r′

Description:

The contents of any register r′ are loaded into any other register r. Note: r, r′ identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register		r, r′
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 1 T states: 4 4 MHz E.T.: 1.0

Condition Bits Affected: None

Example:

If the H register contains the number 8AH, and the E register contains 10H, the instruction

LD H,E

would result in both registers containing 10H.

LD r,n

LoaD

Operation: r ◊ n

Format:

Mnemonic: LD **Operands:** r, n

Object Code:

0	0	r	r	r	1	1	0

n	n	n	n	n	n	n	n

Description:

The eight-bit integer n is loaded into any register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example 1:

After the execution of

LD E,A5H

the contents of register E will be A5H.

Example 2:

After the execution of

LD A,0

register A will contain zero.

LD r,(HL)

LoaD

Operation: r ⇦ (HL)

Format:

Mnemonic: LD **Operands:** r, (HL)

Object Code:

0	1	r	r	r	1	1	0

Description:

The eight-bit contents of memory location (HL) are loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If register pair HL contains the number 75A1H, and memory address 75A1H contains the byte 58H, the execution of

LD C,(HL)

will result in 58H in register C.

LD r,(IX + d)

LoaD

Operation: r ⇦ (IX + d)

Format:

Mnemonic: LD **Operands:** r, (IX + d)

Object Code:

1	1	0	1	1	1	0	1	DD
0	1	r	r	r	1	1	0	
d	d	d	d	d	d	d	d	

Description:

The operand (IX + d) (the contents of the Index Register IX summed with a displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IX contains the number 25AFH, the instruction

LD B,(IX + 19H)

will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

A typical use of this instruction is shown below. If TABL is a location in memory this program will load the first four bytes of the table into registers A, B, C and D.

```
LD    IX, TABL      ; IX points to the table
LD    A, (IX+0)     ; Load first byte
LD    B, (IX+1)     ; Load second byte
LD    C, (IX+2)     ; Load third byte
LD    D, (IX+3)     ; Load fourth byte
```

LD r,(IY + d)

LoaD

Operation: r ⇦ (IY + d)

Format:

Mnemonic: LD **Operands:** r, (IY + d)

Object Code:

1	1	1	1	1	1	0	1	FD
0	1	r	r	r	1	1	0	
d	d	d	d	d	d	d	d	

Description:

The operand (IY + d) (the contents of the Index Register IY summed with a two's complement displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IY contains the number 25AFH, the instruction

LD B,(IY + 19H)

will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

LD (HL),r

LoaD

Operation: (HL) ⇦ r

Format:

Mnemonic: LD **Operands:** (HL), r

Object Code:

0	1	1	1	0	r	r	r

Description:

The contents of register r are loaded into the memory location specified by the contents of the HL register pair. The symbol r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If the contents of register pair HL specify memory location 2146H, and the B register contains the byte 29H, after the execution of

LD (HL),B

memory address 2146H will also contain 29H.

LD (IX + d),r

LoaD

Operation: (IX + d) ⇦ r

Format:

Mnemonic: LD **Operands:** (IX + d), r

Object Code:

1	1	0	1	1	1	0	1	DD
0	1	1	1	0	r	r	r	
d	d	d	d	d	d	d	d	

Description:

The contents of register r are loaded into the memory address specified by the contents of Index Register IX summed with d, a two's complement displacement integer. The symbol r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the C register contains the byte 1CH, and the Index Register IX contains 3100H, then the instruction

LD (IX+6H), C

will perform the sum 3100H+6H and will load 1CH into memory location 3106H.

LD (IY+d),r

LoaD

Operation: (IY+d) ⇦ r

Format:

Mnemonic: LD **Operands:** (IY+d), r

Object Code:

1	1	1	1	1	1	0	1	FD
0	1	1	1	0	r	r	r	
d	d	d	d	d	d	d	d	

Description:

The contents of register r are loaded into the memory address specified by the sum of the contents of the Index Register IY and d, a two's complement displacement integer. The symbol r is specified according to the following table.

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the C register contains the byte 48H, and the Index Register IY contains 2A11H, then the instruction

LD (IY+4H),C

will perform the sum 2A11H+4H, and will load 48H into memory location 2A15.

LD (HL),n

LoaD

Operation: (HL) ◊ n

Format:

Mnemonic: LD **Operands:** (HL), n

Object Code:

0	0	1	1	0	1	1	0	36

n	n	n	n	n	n	n	n

Description:

Integer n is loaded into the memory address specified by the contents of the HL register pair.

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the HL register pair contains 4444H, the instruction

LD (HL),28H

will result in the memory location 4444H containing the byte 28H.

LD (IX + d),n

Load

Operation: (IX + d) ⇦ n

Format:

Mnemonic: LD **Operands:** (IX + d), n

Object Code:

1	1	0	1	1	1	0	1	DD

0	0	1	1	0	1	1	0	36

d	d	d	d	d	d	d	d

n	n	n	n	n	n	n	n

Description:

The n operand is loaded into the memory address specified by the sum of the contents of the Index Register IX and the two's complement displacement operand d.

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IX contains the number 219AH the instruction

LD (IX+5H),5AH

would result in the byte 5AH in the memory address 219FH.
(219FH = 219AH + 5H.)

LD (IY+d),n

LoaD

Operation: (IY+d) ◊ n

Format:

Mnemonic: LD **Operands:** (IY+d), n

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	1	0	36
d	d	d	d	d	d	d	d	
n	n	n	n	n	n	n	n	

Description:

Integer n is loaded into the memory location specified by the contents of the Index Register summed with a two's complement displacement integer d.

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IY contains the number A940H, the instruction

LD (IY+10H),97H

would result in byte 97H in memory location A950H.

LD A,(BC)

LoaD

Operation: A ⇦ (BC)

Format:

Mnemonic: LD **Operands:** A, (BC)

Object Code:

0	0	0	0	1	0	1	0

0A

Description:

The contents of the memory location specified by the contents of the BC register pair are loaded into the Accumulator.

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If the BC register pair contains the number 4747H, and memory address 4747H contains the byte 12H, then the instruction

LD A,(BC)

will result in byte 12H in register A.

LD A,(DE)

LoaD

Operation: A ⇦ (DE)

Format:

Mnemonic: LD **Operands:** A, (DE)

Object Code:

0	0	0	1	1	0	1	0

1A

Description:

The contents of the memory location specified by the register pair DE are loaded into the Accumulator.

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If the DE register pair contains the number 30A2H and memory address 30A2H contains the byte 22H, then the instruction

LD A,(DE)

will result in byte 22H in register A.

LD A,(nn)

LoaD

Operation: A ⇦ (nn)

Format:

Mnemonic: LD **Operands:** A, (nn)

Object Code:

0	0	1	1	1	0	1	0

3A

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Description:

The contents of the memory location specified by the operands nn are loaded into the Accumulator. The first n operand is the low order byte of a two-byte memory address.

M cycles: 4 T states: 13(4,3,3,3) 4 MHz E.T.: 3.25

Condition Bits Affected: None

Example:

If the contents of memory address 8832H is byte 04H, after the instruction

LD A,(8832H)

byte 04H will be in the Accumulator.

LD (BC),A

LoaD

Operation: (BC) ⇦ A

Format:

Mnemonic: LD **Operands:** (BC), A

Object Code:

0	0	0	0	0	0	1	0

02

Description:

The contents of the Accumulator are loaded into the memory location specified by the contents of the register pair BC.

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If the Accumulator contains 7AH and the BC register pair contains 1212H the instruction

LD (BC),A

will result in 7AH being in memory location 1212H.

LD (DE),A

LoaD

Operation: (DE) ⇦ A

Format:

Mnemonic: LD **Operands:** (DE), A

Object Code:

0	0	0	1	0	0	1	0

12

Description:

The contents of the Accumulator are loaded into the memory location specified by the DE register pair.

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

If the contents of register pair DE are 1128H, and the Accumulator contains byte A0H, the instruction

LD (DE),A

will result in A0H being in memory location 1128H.

LD (nn),A

LoaD

Operation: (nn) ⇦ A

Format:

Mnemonic: LD **Operands:** (nn), A

Object Code:

0	0	1	1	0	0	1	0

32

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Description:

The contents of the Accumulator are loaded into the memory address specified by the operands nn. The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 4 T states: 13(4,3,3,3) 4 MHz E.T.: 3.25

Condition Bits Affected: None

Example:

If the contents of the Accumulator are byte D7H, after the execution of
LD (3141H),A
D7H will be in memory location 3141H.

LD A,I

LoaD

Operation: A ⇦ I

Format:

Mnemonic: LD **Operands:** A, I

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	1	0	1	1	1	57

Description:

The contents of the Interrupt Vector Register I are loaded into the Accumulator.

M cycles: 2 T states: 9(4,5) 4 MHz E.T.: 2.25

Condition Bits Affected:

S: Set if I-Reg. is negative; reset otherwise
Z: Set if I-Reg. is zero; reset otherwise
H: Reset
P/V: Contains contents of IFF2
N: Reset
C: Not affected

Note: If an interrupt occurs during execution of this instruction, the Parity flag will contain a 0.

Example:

If the Interrupt Vector Register contains the byte 4AH, after the execution of
LD A,I
the accumulator will also contain 4AH.

LD A,R

LoaD

Operation: A ⇦ R

Format:

Mnemonic: LD **Operands:** A, R

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	1	1	1	1	1	5F

Description:

The contents of Memory Refresh Register R are loaded into the Accumulator.

M cycles: 2 T states: 9(4,5) 4 MHz E.T.: 2.25

Condition Bits Affected:

S: Set if R-Reg. is negative; reset otherwise
Z: Set if R-Reg. is zero; reset otherwise
H: Reset
P/V: Contains contents of IFF2
N: Reset
C: Not affected

Example:

If the Memory Refresh Register contains the byte 4AH, after the execution of

LD A,R

the Accumulator will also contain 4AH.

LD I,A

LoaD

Operation: I ⇦ A

Format:

Mnemonic: LD **Operands:** I, A

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	1	1	47

Description:

The contents of the Accumulator are loaded into the Interrupt Control Vector Register, I.

M cycles: 2 T states: 9(4,5) 4 MHz E.T.: 2.25

Condition Bits Affected: None

Example:

If the Accumulator contains the number 81H, after the instruction

LD I,A

the Interrupt Vector Register will also contain 81H.

LD R,A

LoaD

Operation: R ⇦ A

Format:

Mnemonic: LD **Operands:** R, A

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	1	1	4F

Description:

The contents of the Accumulator are loaded into the Memory Refresh register R.

M cycles: 2 T states: 9(4,5) 4 MHz E.T.: 2.25

Condition Bits Affected: None

Example:

If the Accumulator contains the number B4H, after the instruction

```
LD   R,A
```

the Memory Refresh Register will also contain B4H.

16 Bit Load Group

LD dd,nn

LoaD

Operation: dd ⇦ nn

Format:

Mnemonic: LD **Operands:** dd, nn

Object Code:

0	0	d	d	0	0	0	1

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Description:

The two-byte integer nn is loaded into the dd register pair, where dd defines the BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code is the low order byte.

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

After the execution of

LD HL,5000H

the contents of the HL register pair will be 5000H.

After the execution of

LD BC,2501H

the BC register will contain 2501H.

LD IX,nn

LoaD

Operation: IX ◊ nn

Format:

Mnemonic: LD **Operands:** IX, nn

Object Code:

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	0	1	21
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

Integer nn is loaded into the Index Register IX. The first n operand in the assembled object code above is the low order byte.

M cycles: 4 T states: 14(4,4,3,3) 4 MHz E.T.: 3.50

Condition Bits Affected: None

Example:

After the instruction

LD IX,45A2H

the Index Register will contain integer 45A2H.

LD IY,nn

LoaD

Operation: IY ⇦ nn

Format:

Mnemonic: LD **Operands:** IY, nn

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	0	1	21
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

Integer nn is loaded into the Index Register IY. The first n operand in the assembled object code above is the low order byte.

M cycles: 4 T states: 14(4,4,3,3) 4 MHz E.T.: 3.50

Condition Bits Affected: None

Example:

After the instruction:

LD IY,7733H

the Index Register IY will contain the integer 7733H.

LD HL,(nn)

LoaD

Operation: H ⇦ (nn + 1), L ⇦ (nn)

Format:

Mnemonic: LD **Operands:** HL, (nn)

Object Code:

0	0	1	0	1	0	1	0	2A
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The contents of memory address nn are loaded into the low order portion of register pair HL (register L), and the contents of the next highest memory address (nn + 1) are loaded into the high order portion of HL (register H). The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 5 T states: 16(4,3,3,3,3) 4 MHz E.T.: 4.00

Condition Bits Affected: None

Example:

If address 4545H contains 37H and address 4546H contains A1H, after the instruction

LD HL,(4545H)

the HL register pair will contain A137H.

LD dd,(nn) LoaD

Operation: $dd_H \Leftarrow (nn+1)$, $dd_L \Leftarrow (nn)$

Format:

Mnemonic: LD **Operands:** dd, (nn)

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	d	d	1	0	1	1	
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The contents of address nn are loaded into the low order portion of register pair dd, and the contents of the next highest memory address (nn + 1) are loaded into the high order portion of dd. Register pair dd defines BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code above is the low order byte of (nn).

M cycles: 6 T states: 20(4,4,3,3,3,3) 4 MHz E.T.: 5.00

Condition Bits Affected: None

Example 1:

If Address 2130H contains 65H and address 2131M contains 78H after the instruction

LD BC,(2130H)

the BC register pair will contain 7865H.

Example 2:

If address FFFE contains 01H and address FFFF contains 02H, then after the instruction

LD SP,(0FFFEH)

the SP will contain 0201H.

LD IX,(nn)

LoaD

Operation: $IX_H \Leftarrow (nn+1)$, $IX_L \Leftarrow (nn)$

Format:

Mnemonic: LD **Operands:** IX, (nn)

Object Code:

1	1	0	1	1	1	0	1	DD
0	0	1	0	1	0	1	0	2A
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The contents of the address nn are loaded into the low order portion of Index Register IX, and the contents of the next highest memory address (nn + 1) are loaded into the high order portion of IX. The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 6 T states: 20(4,4,3,3,3,3) 4 MHz E.T.: 5.00

Condition Bits Affected: None

Example:

If address 6066H contains 92H and address 6067H contains DAH, after the instruction

LD IX,(6066H)

the Index Register IX will contain DA92H.

LD IY,(nn)

LoaD

Operation: $IY_H \Leftarrow (nn+1)$, $IY_L \Leftarrow (nn)$

Format:

Mnemonic: LD **Operands:** IY, (nn)

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	0	1	0	1	0	2A
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The contents of address nn are loaded into the low order portion of Index Register IY, and the contents of the next highest memory address (nn + 1) are loaded into the high order portion of IY. The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 6 T states: 20(4,4,3,3,3,3) 4 MHz E.T.: 5.00

Condition Bits Affected: None

Example:

If address 6666H contains 92H and address 6667H contains DAH, after the instruction

LD IY,(6666H)

the Index Register IY will contain DA92H.

LD (nn),HL

LoaD

Operation: (nn + 1) ⇦ H, (nn) ⇦ L

Format:

Mnemonic: LD **Operands:** (nn), HL

Object Code:

0	0	1	0	0	0	1	0	22
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The contents of the low order portion of register pair HL (register L) are loaded into memory address nn, and the contents of the high order portion of HL (register H) are loaded into the next highest memory address (nn + 1). The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 5 T states: 16(4,3,3,3,3) 4 MHz E.T.: 4.00

Condition Bits Affected: None

Example 1:

If the content of register pair HL is 483AH, after the instruction

LD (B229H),HL

address B229H will contain 3AH, and address B22AH will contain 48H.

Example 2:

If the register pair HL contains 504AH, then after the instruction

LD (PLACE),HL

the address PLACE will contain 4AH and address PLACE + 1 will contain 50H.

Note: PLACE is a label which must be defined elsewhere in the program.

LD (nn),dd

LoaD

Operation: $(nn + 1) \leftarrow dd_H$, $(nn) \leftarrow dd_L$

Format:

Mnemonic: LD **Operands:** (nn), dd

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	d	d	0	0	1	1	
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The low order byte of register pair dd is loaded into memory address (nn); the upper order byte is loaded into memory address (nn + 1). Register pair dd defines either BC, DE, HL, or SP, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code is the low order byte of a two byte memory address.

M cycles: 6 T states: 20(4,4,3,3,3,3) 4 MHz E.T.: 5.00

Condition Bits Affected: None

Example:

If register pair BC contains the number 4644H, the instruction

LD (1000H),BC

will result in 44H in memory location 1000H, and 46H in memory location 1001H.

LD (nn),IX

LoaD

Operation: $(nn + 1) \Leftarrow IX_H$, $(nn) \Leftarrow IX_L$

Format:

Mnemonic: LD **Operands:** (nn), IX

Object Code:

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	1	0	22
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The low order byte in Index Register IX is loaded into memory address nn; the upper order byte is loaded into the next highest address (nn + 1). The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 6 T states: 20(4,4,3,3,3,3) 4 MHz E.T.: 5.00

Condition Bits Affected: None

Example:

If the Index Register IX contains 5A30H, after the instruction

LD (4392H),IX

memory location 4392H will contain number 30H and location 4393H will contain 5AH.

LD (nn),IY

LoaD

Operation: $(nn+1) \leftarrow IY_H$, $(nn) \leftarrow IY_L$

Format:

Mnemonic: LD **Operands:** (nn), IY

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	1	0	22
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Description:

The low order byte in Index Register IY is loaded into memory address nn; the upper order byte is loaded into memory location (nn + 1). The first n operand in the assembled object code above is the low order byte of nn.

M cycles: 6 T states: 20(4,4,3,3,3,3) 4 MHz E.T.: 5.00

Condition Bits Affected: None

Example:

If the Index Register IY contains 4174H after the instruction

LD 8838H,IY

memory location 8838H will contain number 74H and memory location 8839H will contain 41H.

LD SP,HL

LoaD

Operation: SP ◊ HL

Format:

Mnemonic: LD **Operands:** SP, HL

Object Code:

1	1	1	1	1	0	0	1	F9

Description:

The contents of the register pair HL are loaded into the Stack Pointer SP.

M cycles: 1 T states: 6 4 MHz E.T.: 1.50

Condition Bits Affected: None

Example:

If the register pair HL contains 442EH, after the instruction

LD SP,HL

the Stack Pointer will also contain 442EH.

LD SP,IX

LoaD

Operation: SP ⇄ IX

Format:

Mnemonic: LD **Operands:** SP, IX

Object Code:

1	1	0	1	1	1	0	1	DD
1	1	1	1	1	0	0	1	F9

Description:

The two-byte contents of Index Register IX are loaded into the Stack Pointer SP.

M cycles: 2 T states: 10(4,6) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Index Register IX are 98DAH, after the instruction

LD SP,IX

the contents of the Stack Pointer will also be 98DAH.

LD SP,IY

LoaD

Operation: SP ⇦ IY

Format:

Mnemonic: LD **Operands:** SP, IY

Object Code:

1	1	1	1	1	1	0	1	FD

1	1	1	1	1	0	0	1	F9

Description:

The two byte contents of Index Register IY are loaded into the Stack Pointer SP.

M cycles: 2 T states: 10(4,6) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If Index Register IY contains the integer A227H, after the instruction
LD SP,IY
the Stack Pointer will also contain A227H.

PUSH qq

Operation: $(SP-2) \Leftarrow qq_L$, $(SP-1) \Leftarrow qq_H$

Format:

Mnemonic: PUSH **Operands:** qq

Object Code:

1	1	q	q	0	1	0	1

Description:

The contents of the register pair qq are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current ''top'' of the Stack. This instruction first decrements the SP and loads the high order byte of register pair qq into the memory address now specified by the SP, then decrements the SP again and loads the low order byte of qq into the memory location corresponding to this new address in the SP. The operand qq means register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	qq
BC	00
DE	01
HL	10
AF	11

M cycles: 3 T states: 11(5,3,3) 4 MHz E.T.: 2.75

Condition Bits Affected: None

Example:

If the AF register pair contains 2233H and the Stack Pointer contains 1007H, after the instruction

PUSH AF

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H. In other words the number from register pair AF is now on the top of the stack, and the stack pointer is pointing to it.

Before:

Register AF	Address	Stack
2233	1007	FF
	1008	35

Stack Pointer

1007

After: PUSH AF

Register AF	Address	Stack
2233	1005	33
	1006	22
	1007	FF
	1008	35

Stack Pointer

1005

PUSH IX

Operation: $(SP-2) \leftarrow IX_L$, $(SP-1) \leftarrow IX_H$

Format:

Mnemonic: PUSH **Operands:** IX

Object Code:

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	1	0	1	E5

Description:

The contents of the Index Register IX are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IX into the memory address now specified by the SP, then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M cycles: 3 T states: 15(4,5,3,3) 4 MHz E.T.: 3.75

Condition Bits Affected: None

Example:

If the Index Register IX contains 2233H and the Stack Pointer contains 1007H, after the instruction

PUSH IX

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H. The number from the IX register pair is now on the top of the stack.

Before:

Register IX	Address	Stack
2233	1007	FF
	1008	35

Stack Pointer

1007

After: PUSH IX

Register IX	Address	Stack
2233	1005	33
	1006	22
	1007	FF
	1008	35

Stack Pointer

1005

PUSH IY

Operation: $(SP-2) \Leftarrow IY_L$, $(SP-1) \Leftarrow IY_H$

Format:

Mnemonic: PUSH **Operands:** IY

Object Code:

1	1	1	1	1	1	0	1	FD

1	1	1	0	0	1	0	1	E5

Description:

The contents of the Index Register IY are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current ''top'' of the Stack. This instruction first decrements the SP and loads the high order byte of IY into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M cycles: 4 T states: 15(4,5,3,3) 4 MHz E.T.: 3.75

Condition Bits Affected: None

Example:

If the Index Register IY contains 2233H and the Stack Pointer contains 1007H, after the instruction

PUSH IY

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H. The number from register pair IY is now on the top of the stack.

Before:

Register IY	Address	Stack
2233	1007	FF
	1008	35

Stack Pointer

1007

After: PUSH IY

Register IY	Address	Stack
2233	1005	33
	1006	22
	1007	FF
	1008	35

Stack Pointer

1005

POP qq

Operation: $qq_H \leftarrow (SP+1),\ qq_L \leftarrow (SP)$

Format:

Mnemonic: POP **Operands:** qq

Object Code:

1	1	q	q	0	0	0	1

Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into register pair qq. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of qq, the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of qq and the SP is now incremented again. The operand qq defines register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	r
BC	00
DE	01
HL	10
AF	11

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

POP HL

will result in register pair HL containing 3355H, and the Stack Pointer containing 1002H. In other words register pair HL contains the number which was on the top of the stack, and the stack pointer is pointing to the current top of the stack.

Before:

Register HL	Address	Stack
2233	1000	55
	1001	33
	1002	A4
	1003	62

Stack Pointer

1000

After: POP HL

Register HL	Address	Stack
3355	1002	A4
	1003	62

Stack Pointer

1002

POP IX

Operation: $IX_H \leftarrow (SP+1)$, $IX_L \leftarrow (SP)$

Format:

Mnemonic: POP **Operands:** IX

Object Code:

1	1	0	1	1	1	0	1	DD

1	1	1	0	0	0	0	1	E1

Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IX. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IX the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IX. The SP is now incremented again.

M cycles: 4 T states: 14(4,4,3,3) 4 MHz E.T.: 3.50

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

POP IX

will result in the Index Register IX containing 3355H, and the Stack Pointer containing 1002H. Register pair IX contains the number which used to be on the top of the stack.

Before:

Register IX	Address	Stack
24F9	1000	55
	1001	33
	1002	A4
	1003	62

Stack Pointer

1000

After: POP IX

Register IX	Address	Stack
3355	1002	A4
	1003	62

Stack Pointer
1002

POP IY

Operation: $IY_H \Leftarrow (SP+1), IY_L \Leftarrow (SP)$

Format:

Mnemonic: POP **Operands:** IY

Object Code:

1	1	1	1	1	1	0	1	FD

1	1	1	0	0	0	0	1	E1

Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IY. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IY the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IY. The SP is now incremented again.

M cycles: 4 T states: 14(4,4,3,3) 4 MHz E.T.: 3.50

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

POP IY

will result in Index Register IY containing 3355H, and the Stack Pointer containing 1002H. Register pair IY contains the number which used to be on the top of the stack.

Before:

Register IY	Address	Stack
24F9	1000	55
	1001	33
	1002	A4
	1003	62

Stack Pointer

1000

After: POP IY

Register IY	Address	Stack
3355	1002	A4
	1003	62

Stack Pointer

1002

Exchange, Block Transfer and Search Group

EX DE,HL

EXchange

Operation: DE ⇔ HL

Format:

Mnemonic: EX **Operands:** DE, HL

Object Code:

1	1	1	0	1	0	1	1

EB

Description:

The two-byte contents of register pairs DE and HL are exchanged.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

Example:

If the content of register pair DE is the number 2822H, and the content of the register pair HL is number 499AH, after the instruction

EX DE,HL

the content of register pair DE will be 499AH and the content of register pair HL will be 2822H.

EX AF,AF′

EXchange

Operation: AF ⇔ AF′

Format:

Mnemonic: EX **Operands:** AF, AF′

Object Code:

0	0	0	0	1	0	0	0

08

Description:

The two-byte contents of the register pairs AF and AF′ are exchanged. (Note: register pair AF′ consists of registers A′ and F′.)

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

Example:

If the content of register pair AF is number 9900H, and the content of register pair AF′ is number 5944H, after the instruction

EX AF,AF′

the contents of AF will be 5944H, and the contents of AF will be 9900H.

EXX

EXchange

Operation: (BC) ⇔ (BC′), (DE) ⇔ (DE′), (HL) ⇔ (HL′)

Format:

Mnemonic: EXX **Operands:**

Object Code:

1	1	0	1	1	0	0	1

D9

Description:

Each two-byte value in register pairs BC, DE, and HL is exchanged with the two-byte value in BC′, DE′, and HL′, respectively.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

Example 1:

If the contents of register pairs BC, DE, and HL are the numbers 445AH, 3DA2H, and 8859H, respectively, and the contents of register pairs BC′, DE′, and HL′ are 0988H, 9300H, and 00E7H, respectively, after the instruction

EXX

the contents of the register pairs will be as follows: BC: 0988H; DE: 9300H; HL: 00E7H; BC′: 445AH; DE′: 3DA2H; and HL′: 8859H.

Example 2:

If the contents of the registers are as shown:

BC	:	1111H
DE	:	2222H
HL	:	3333H
BC′	:	4444H
DE′	:	5555H
HL′	:	6666H

Then after an EXX instruction the registers will contain:

BC	:	4444H
DE	:	5555H
HL	:	6666H
BC′	:	1111H
DE′	:	2222H
HL′	:	3333H

EX (SP), HL

EXchange

Operation: H ⇔ (SP + 1), L ⇔ (SP)

Format:

Mnemonic: EX **Operands:** (SP),HL

Object Code:

1	1	1	0	0	0	1	1

E3

Description:

The low order byte contained in register pair HL is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of HL is exchanged with the next highest memory address (SP + 1).

M cycles: 5 T states: 19(4,3,4,3,5) 4 MHz E.T.: 4.75

Condition Bits Affected: None

Example:

If the HL register pair contains 7012H, the SP register pair contains 8856H, the memory location 8856H contains the byte 11H, and the memory location 8857H contains the byte 22H, then the instruction

EX (SP),HL

will result in the HL register pair containing number 2211H, memory location 8856H containing the byte 12H, the memory location 8857H containing the byte 70H and the Stack Pointer containing 8856H.

Before:

Register HL	Address	Stack
7012	8856	11
	8857	22
	8858	

Stack Pointer

8856

After:

Register HL	Address	Stack
2211	8856	12
	8857	70
	8858	

Stack Pointer

8856

EX (SP),IX

EXchange

Operation: $IX_H \Leftrightarrow (SP+1)$, $IX_L \Leftrightarrow (SP)$

Format:

Mnemonic: EX **Operands:** (SP), IX

Object Code:

1	1	0	1	1	1	0	1	DD

1	1	1	0	0	0	1	1	E3

Description:

The low order byte in Index Register IX is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IX is exchanged with the next highest memory address (SP + 1).

Condition Bits Affected: None

Example:

If the Index Register IX contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

EX (SP),IX

will result in the IX register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H and the Stack Pointer containing 0100H.

Before:

Register IX	Address	Stack
3988	0100	90
	0101	48

Stack Pointer

0100

After:

Register IX	Address	Stack
4890	0100	88
	0101	39

Stack Pointer

0100

EX (SP),IY

EXchange

Operation: $IY_H \Leftrightarrow (SP + 1)$, $IY_L \Leftrightarrow (SP)$

Format:

Mnemonic: EX **Operands:** (SP), IY

Object Code:

1	1	1	1	1	1	0	1	FD
1	1	1	0	0	0	1	1	E3

Description:

The low order byte in Index Register IY is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IY is exchanged with the next highest memory address (SP + 1).

M cycles: 6 T states: 23(4,4,3,4,3,5) 4 MHz E.T.: 5.75

Condition Bits Affected: None

Example:

If the Index Register IY contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

EX (SP),IY

will result in the IY register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H, and the Stack Pointer containing 0100H.

Before:

Register IY	Address	Stack
3988	0100	90
	0101	48

Stack Pointer

0100

After:

Register IY	Address	Stack
4890	0100	88
	0101	39

Stack Pointer

0100

LDI

LoaD & Increment

Operation: (DE) ◊ (HL), DE ◊ DE + 1, HL ◊ HL + 1, BC ◊ BC − 1

Format:

Mnemonic: LDI **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	0	0	AO

Description:

A byte of data is transferred from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented.

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Set if BC − 1 ≠ 0; reset otherwise
N: Reset
C: Not affected

Example 1:

If the HL register pair contains 1111H, memory location 1111H contains the byte 88H, the DE register pair contains 2222H, the memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

LDI

will result in the following contents in register pairs and memory addresses:

HL : 1112H
(1111H) : 88H
DE : 2223H
(2222H) : 88H
BC : 6H

and the condition Bits will be:

		0	1	0	
S	Z	H	P/V	N	C

Example 2:

If the contents of registers and memory are as shown:

```
HL      :  7C00H
(7C00)  :  FFH
DE      :  3C00H
(3C00)  :  00H
BC      :  1H
```

Then after an LDI instruction the registers and memory will contain the following:

```
HL      :  7C01H
(7C00)  :  FFH
DE      :  3C01H
(3C00)  :  FFH
BC      0H
```

and the condition bits will be:

		0	0	0	
S	Z	H	P/V	N	C

Example 3:

The following program will move 80 consecutive bytes from BUF1 to BUF2:

```
LD      HL, BUF1
LD      DE, BUF2
LD      BC, 80
LOOP    LDI
JP      NZ, LOOP
```

LDIR

LoaD Increment & Repeat

Operation: $(DE) \leftarrow (HL),\ DE \leftarrow DE + 1,\ HL \leftarrow HL + 1,\ BC \leftarrow BC - 1$

Format:

Mnemonic: LDIR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	0	B0

Description:

This two-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero the program counter (PC) is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes. Also, interrupts will be recognized after each data transfer.

For BC ≠ 0:

M cycles: 5 T states: 21(4,4,3,5,5) 4 MHz E.T.: 5.25

For BC = 0:

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Reset
N: Reset
C: Not affected

Example:

If the HL register pair contains 1111H, the DE register pair contains 2222H, the BC register pair contains 0003H, and memory locations have these contents:

(1111H)	:	88H	(2222H)	:	66H
(1112H)	:	36H	(2223H)	:	59H
(1113H)	:	A5H	(2224H)	:	C5H

then after the execution of

LDIR

the contents of register pairs and memory locations will be:

HL	:	1114H			
DE	:	2225H			
BC	:	0000H			
(1111H)	:	88H	(2222H)	:	88H
(1112H)	:	36H	(2223H)	:	36H
(1113H)	:	A5H	(2224H)	:	A5H

and the H, P/V, and N flags are all zero.

LDD

LoaD Decrement

Operation: (DE) ◊ (HL), DE ◊ DE − 1, HL ◊ HL − 1, BC ◊ BC − 1

Format:

Mnemonic: LDD **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED

1	0	1	0	1	0	0	0	A8

Description:

This two-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these register pairs, including the BC (Byte Counter) register pair, are decremented.

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Set if BC − 1 ≠ 0; reset otherwise
N:	Reset
C:	Not affected

Example 1:

If the HL register pair contains 1111H, memory location 1111H contains the byte 88H, the DE register pair contains 2222H, memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

LDD

will result in the following contents in register pairs and memory addresses:

HL : 1110H
(1111H) : 88H
DE : 2221H
(2222H) : 88H
BC : 6H

and the condition bits will be:

		0	1	0	
S	Z	H	P/V	N	C

Example 2:

If the contents of registers and memory are as shown:

HL : 7CFFH
(7CFF) : 3CH
DE : 3CFFH
(3CFF) : 00H
BC : 1H

Then after a LDD instruction the registers and memory will contain the following:

HL : 7CFEH
(7CFF) : 3CH
DE : 3CFEH
(3CFF) : 3CH
BC : 0H

and the condition bits will be:

		0	0	0	
S	Z	H	P/V	N	C

LDDR

LoaD Decrement & Repeat

Operation: (DE) ◊ (HL), DE ◊ DE – 1, HL ◊ HL – 1, BC ◊ BC – 1

Format:

Mnemonic: LDDR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	0	B8

Description:

This two-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these registers as well as the BC (Byte Counter) are decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero, the program counter (PC) is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes. Also, interrupts will be recognized after each data transfer.

For BC ≠ 0:

M cycles: 5 T states: 21(4,4,3,5,5) 4 MHz E.T.: 5.25

For BC = 0:

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Reset
N: Reset
C: Not affected

Example:

If the HL register pair contains 1114H, the DE register pair contains 2225H, the BC register pair contains 0003H, and memory locations have these contents:

(1114H)	:	A5H	(2225H)	:	C5H
(1113H)	:	36H	(2224H)	:	59H
(1112H)	:	88H	(2223H)	:	66H

then after the execution of

LDDR

the contents of register pairs and memory locations will be:

HL	:	1111H			
DE	:	2222H			
BC	:	0000H			
(1114H)	:	A5H	(2225H)	:	A5H
(1113H)	:	36H	(2224H)	:	36H
(1112H)	:	88H	(2223H)	:	88H

and the H, P/V, and N flags are all zero.

CPI

ComPare & Increment

Operation: A − (HL), HL ◊ HL + 1, BC ◊ BC − 1

Format:

Mnemonic: CPI **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED

1	0	1	0	0	0	0	1	A1

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, the Z condition bit is set. Then HL is incremented and the Byte Counter (register pair BC) is decremented.

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if A = (HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Reset if BC becomes 0; set otherwise
N:	Set
C:	Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H, then after the execution of

CPI

the Byte Counter will contain 0000H, the HL register pair will contain 1112H, the Z flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

If the contents of memory and registers are as shown

```
HL      : 8A00H
(8A00H) : 6DH
A       : 75H
BC      : 5H
```

Then during the execution of a CPI instruction the Arithmetic and Logic Unit will do the following subtraction:

```
                    Borrow needed here
                 ⇩

   75H = 0111  0101
 - 6DH = 0110  1101
 -------------------
    8H = 0000  1000
```

After CPI is executed registers and memory will contain the following:

```
HL      : 8A01H
(8A00H) : 6DH
A       : 75H
BC      : 4H
```

and the condition bits would be:

	S	Z	H	P/V	N	C	
	0	0	1	1	1	1	
result positive	⇩						not affected
match not found		⇩					always set
borrow from bit 4			⇩				BC not zero

Example 3:

The following program is used to verify that the contents of two 80-byte buffers are identical. Each time a mismatch is found the program calls a subroutine called ERROR.

```
STRT    LD      HL, BUF1
        LD      DE, BUF2
        LD      BC, 80
LOOP    LD      A, (DE)
        CPI
        CALL    NZ, ERROR
        INC     DE
        JR      PO, LOOP
END
```

CPIR

ComPare Increment & Repeat

Operation: $A - (HL), HL \Leftarrow HL + 1, BC \Leftarrow BC - 1$

Format:

Mnemonic: CPIR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	1	B1

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, the Z condition bit is set. The HL is incremented and the Byte Counter (register pair BC) is decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A ≠ (HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero before the execution, the instruction will loop through 64K bytes, if no match is found. Also, interrupts will be recognized after each data comparison.

For BC ≠ 0 and A ≠ (HL):

M cycles: 5 T states: 21(4,4,3,5,5) 4 MHz E.T.: 5.25

For BC ≠ 0 or A = (HL):

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if A = (HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Reset if BC becomes 0; set otherwise
N:	Set
C:	Not affected

Example:

If the HL register pair contains 1111H, the Accumulator (Register A) contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1111H) : 52H
(1112H) : 00H
(1113H) : F3H

then after the execution of

CPIR

the contents of register pair HL will be 1114H, and the contents of the Byte Counter will be 0004H. Since BC ≠ 0, the P/V flag is still set. This means that it did not search through the whole block before the instruction stopped. Since a match was found, the Z flag is set.

The following program uses the CPIR instruction to count the number of nulls (00H) found in an 80-byte buffer. The count is kept in register E.

```
STRT    LD      BC, 80
        LD      HL, BUFF
        LD      A, 0
        LD      E, 0
LOOP    CPIR
        JR      NZ, FOO
        INC     E
FOO     JP      PE, LOOP
END
```

CPD

ComPare & Decrement

Operation: A − (HL), HL ◊ HL − 1, BC ◊ BC − 1

Format:

Mnemonic: CPD **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	0	1	A9

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, the Z condition bit is set. The HL and the Byte Counter (register pair BC) are decremented.

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if A = (HL); reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Reset if BC becomes zero; set otherwise
N: Set
C: Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H, then after the execution of

CPD

the Byte Counter will contain 0000H, the HL register pair will contain 1110H, the Z flag in the F register will be set and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

Since the CPD instruction decrements HL, it is used to search through memory from high to low addresses. Otherwise it is similar to the CPI instruction.

CPDR ComPare Decrement & Repeat

Operation: A − (HL), HL ◊ HL − 1, BC ◊ BC − 1

Format:

Mnemonic: CPDR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	1	B9

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, the Z condition bit is set. The HL and BC (Byte Counter) register pairs are decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A ≠ (HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes, if no match is found. Also, interrupts will be recognized after each data comparison.

For BC ≠ 0 and A ≠ (HL):

M cycles: 5 T states: 21(4,4,3,5,5) 4 MHz E.T.: 5.25

For BC = 0 or A = (HL):

M cycles: 4 T states: 16(4,4,3,5) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if A = (HL), reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Reset if BC becomes zero; set otherwise
N: Set
C: Not affected

Example:

If the HL register pair contains 1118H, the Accumulator contains F3H, the Byte Counter contains 0003H, and memory locations have these contents:

(1118H) : 52H
(1117H) : 00H
(1116H) : F3H

then after the execution of

CPDR

the contents of register pair HL will be 1115H, the contents of the Byte Counter will be 0000H, the P/V flag in the F register will be reset, and the Z flag in the F register will be set.

8 Bit Arithmetic and Logical Group

ADD A,r

Operation: A ⇦ A + r

Format:

Mnemonic: ADD **Operands:** A, r

Object Code:

1	0	0	0	0	r	r	r

Description:

The contents of register r are added to the contents of the Accumulator, and the result is stored in the Accumulator. The symbol r identifies the registers A, B, C, D, E, H or L assembled as follows in the object code:

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are 44H, and the contents of register C are 11H, after the execution of

ADD A,C

the contents of the Accumulator will be 55H. See Appendix K for more details of condition bits affected.

ADD A,n

Operation: A ◊ A + n

Format:

Mnemonic: ADD **Operands:** A, n

Object Code:

1	1	0	0	0	1	1	0

C6

n	n	n	n	n	n	n	n

Description:

The integer n is added to the contents of the Accumulator and the results are stored in the Accumulator.

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are 23H, after the execution of

ADD A,33H

the contents of the Accumulator will be 56H.

ADD A,(HL)

Operation: A ◊ A + (HL)

Format:

Mnemonic: ADD **Operands:** A, (HL)

Object Code:

1	0	0	0	0	1	1	0

86

Description:

The byte at the memory address specified by the contents of the HL register pair is added to the contents of the Accumulator and the result is stored in the Accumulator.

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are A0H, and the content of the register pair HL is 2323H, and memory location 2323H contains byte 08H, after the execution of

ADD A,(HL)

the Accumulator will contain A8H.

ADD A,(IX+d)

Operation: A ◊ A + (IX+d)

Format:

Mnemonic: ADD **Operands:** A, (IX+d)

Object Code:

1	1	0	1	1	1	0	1	DD
1	0	0	0	0	1	1	0	86
d	d	d	d	d	d	d	d	

Description:

The contents of the Index Register (register pair IX) is added to a two's complement displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contents are 11H, the Index Register IX contains 1000H, and if the content of memory location 1005H is 22H, after the execution of

ADD A,(IX+5H)

the contents of the Accumulator will be 33H.

ADD A,(IY+d)

Operation: A◊A+(IY+d)

Format:

Mnemonic: ADD **Operands:** A, (IY+d)

Object Code:

1	1	1	1	1	1	0	1	FD
1	0	0	0	0	1	1	0	86
d	d	d	d	d	d	d	d	

Description:

The contents of the Index Register (register pair IY) is added to the displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M cycles: 5 T states: 19(4,4,3,5,3) 4 MHz E.T.: 4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contents are 11H, the Index Register pair IY contains 1000H, and if the content of memory location 1005H is 22H, after the execution of

ADD A,(IY+5H)

the contents of the Accumulator will be 33H.

ADC A,S

ADd with Carry

Operation: A ◊ A + s + CY

Format:

Mnemonic: ADC **Operands:** A, s

The s operand is any of r, n, (HL), (IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

ADC A, r	1	0	0	0	1	r	r	r	
ADC A, n	1	1	0	0	1	1	1	0	CE
	n	n	n	n	n	n	n	n	
ADC A, (HL)	1	0	0	0	1	1	1	0	8E
ADC A, (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	0	0	1	1	1	0	8E
	d	d	d	d	d	d	d	d	
ADC A, (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	0	0	1	1	1	0	8E
	d	d	d	d	d	d	d	d	

r identifies registers A, B, C, D, E, H, or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

The s operand, along with the Carry Flag (''C'' in the F register) is added to the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T. in μs
ADC A, r	1	4	1.00
ADC A, n	2	7(4,3)	1.75
ADC A, (HL)	2	7(4,3)	1.75
ADC A, (IX+d)	5	19(4,4,3,5,3)	4.75
ADC A, (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 7; reset otherwise

Example 1:

If the Carry Flag is set, the Accumulator contains 16H, the HL register pair contains 6666H, and address 6666H contains 10H, after the execution of

ADC A, (HL)

the Accumulator will contain 27H.

Example 2:

If the Carry Flag is set, the Accumulator contains 30H, and register C contains 0 5H, then after the execution of

ADC A, C

the Accumulator will contain 36H.

SUB s

SUBtract

Operation: A ⇦ A − s

Format:

Mnemonic: SUB **Operands:** s

The s operand is any of r, n, (HL), (IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

SUB r	1	0	0	1	0	r	r	r	
SUB n	1	1	0	1	0	1	1	0	D6
	n	n	n	n	n	n	n	n	
SUB (HL)	1	0	0	1	0	1	1	0	96
SUB (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	0	1	0	1	1	0	96
	d	d	d	d	d	d	d	d	
SUB (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	0	1	0	1	1	0	96
	d	d	d	d	d	d	d	d	

r identifies registers A, B, C, D, E, H or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

The s operand is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T. in μs
SUB r	1	4	1.00
SUB n	2	7(4,3)	1.75
SUB (HL)	2	7(4,3)	1.75
SUB (IX+d)	5	19(4,4,3,5,3)	4.75
SUB (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Set if overflow; reset otherwise
N: Set
C: Set if borrow; reset otherwise

Example:

If the Accumulator contains 29H and register D contains 11H, after the execution of

SUB D

the Accumulator will contain 18H.

SBC A,s

SuBtract with borrow (Carry)

Operation: A ◊ A − s − CY

Format:

Mnemonic: SBC **Operands:** A, s

The s operand is any of r, n, (HL), (IX+d) or (IY+d) as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

SBC A, r	1	0	0	1	1	r	r	r	
SBC A, n	1	1	0	1	1	1	1	0	DE
	n	n	n	n	n	n	n	n	

SBC A, (HL)	1	0	0	1	1	1	1	0	9E
SBC A, (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	0	1	1	1	1	0	9E
	d	d	d	d	d	d	d	d	
SBC A,(IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	0	1	1	1	1	0	9E
	d	d	d	d	d	d	d	d	

r identifies registers A, B, C, D, E, H, or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

The s operand, along with the Carry Flag (''C'' in the F register) is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T. in μs
SBC A, r	1	4	1.00
SBC A, n	2	7(4,3)	1.75
SBC A, (HL)	2	7(4,3)	1.75
SBC A, (IX+d)	5	19(4,4,3,5,3)	4.75
SBC A, (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Set if overflow; reset otherwise
N: Set
C: Set if borrow; reset otherwise

Example 1:

If the Carry Flag is set, the Accumulator contains 16H, the HL register pair contains 3433H, and address 3433H contains 05H, after the execution of

SBC A,(HL)

the Accumulator will contain 10H.

Example 2:

If the Carry Flag is set, the Accumulator contains 21H and register B contains 0, then after the execution of

SBC A,B

the Accumulator contains 20H.

AND s

Operation: A ⇦ A ⌔ s

Format:

Mnemonic: AND **Operands:** s

The s operand is any of r, n, (HL), (IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

AND r	1	0	1	0	0	r	r	r	
AND n	1	1	1	0	0	1	1	0	E6
	n	n	n	n	n	n	n	n	
AND (HL)	1	0	1	0	0	1	1	0	A6
AND (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	0	0	1	1	0	A6
	d	d	d	d	d	d	d	d	
AND (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	0	0	1	1	0	A6
	d	d	d	d	d	d	d	d	

r identifies register A, B, C, D, E, H or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

A logical AND operation, Bit by Bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T. in μs
AND r	1	4	1.00
AND n	2	7(4,3)	1.75
AND (HL)	2	7(4,3)	1.75
AND (IX+d)	5	19(4,4,3,5,3)	4.75
AND (IX+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Reset

Table of AND Values:

IF A	IF B	Then A (After)
0	0	0
0	1	0
1	0	0
1	1	1

Example:

If the B register contains 7BH (01111011) and the Accumulator contains C3H (11000011), after the execution of

AND B

the Accumulator will contain 43H (01000011).

OR s

Operation: A ⇦ A ⊽ s

Format:

Mnemonic: OR **Operands:** s

The s operand is any of r, n, (HL), (IX+d), or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

OR r	1	0	1	1	0	r	r	r	
OR n	1	1	1	1	0	1	1	0	F6
	n	n	n	n	n	n	n	n	
OR (HL)	1	0	1	1	0	1	1	0	B6
OR (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	1	0	1	1	0	B6
	d	d	d	d	d	d	d	d	
OR (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	1	0	1	1	0	B6
	d	d	d	d	d	d	d	d	

r identifies register A, B, C, D, E, H or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

A logical OR operation, Bit by Bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T. in μs
OR r	1	4	1.00
OR n	2	7(4,3)	1.75
OR (HL)	2	7(4,3)	1.75
OR (IX+d)	5	19(4,4,3,5,3)	4.75
OR (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Reset

Table of OR Values:

IF A	IF B	Then A (After)
0	0	0
0	1	1
1	0	1
1	1	1

Example:

If the H register contains 48H (01001000) and the Accumulator contains 12H (00010010), after the execution of

OR H

the Accumulator will contain 5AH (01011010).

XOR s

eXclusive OR

Operation: $A \leftarrow A \oplus s$

Format:

Mnemonic: XOR **Operands:** s

The s operand is any of r, n, (HL), (IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

XOR r	1	0	1	0	1	r	r	r	
XOR n	1	1	1	0	1	1	1	0	EE
	n	n	n	n	n	n	n	n	
XOR (HL)	1	0	1	0	1	1	1	0	AE
XOR (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	0	1	1	1	0	AE
	d	d	d	d	d	d	d	d	
XOR (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	0	1	1	1	0	AE
	d	d	d	d	d	d	d	d	

r identifies registers A, B, C, D, E, H or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

A logical exclusive-OR operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T. in μs
XOR r	1	4	1.00
XOR n	2	7(4,3)	1.75
XOR (HL)	2	7(4,3)	1.75
XOR (IX+d)	5	19(4,4,3,5,3)	4.75
XOR (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Reset

Table of XOR Values:

IF A	IF B	Then A (After)
0	0	0
0	1	1
1	0	1
1	1	0

Note: in Table above that any two like numbers will result in zero.

Example 1:

If the Accumulator contains 96H (10010110), after the execution of

XOR 5DH (Note: 5DH = 01011101)

the Accumulator will contain CBH (11001011).

Example 2:

The instruction

XOR A

will zero the Accumulator.

CP s

ComPare

Operation: A − s

Format:

Mnemonic: CP **Operands:** s

The s operand is any of r, n, (HL), (IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

CP r	1	0	1	1	1	r	r	r	
CP n	1	1	1	1	1	1	1	0	FE
	n	n	n	n	n	n	n	n	
CP (HL)	1	0	1	1	1	1	1	0	BE
CP (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	1	1	1	1	0	BE
	d	d	d	d	d	d	d	d	
CP (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	1	1	1	1	0	BE
	d	d	d	d	d	d	d	d	

r identifies register A, B, C, D, E, H or L assembled as follows in the object code field above:

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

The contents of the s operand are compared with the contents of the Accumulator. If there is a true compare, a flag is set.

Instruction	M Cycles	T States	4 MHz E.T. in μs
CP r	1	4	1.00
CP n	2	7(4,3)	1.75
CP (HL)	2	7(4,3)	1.75
CP (IX+d)	5	19(4,4,3,5,3)	4.75
CP (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Set
C:	Set if borrow in Bit 7; reset otherwise

Example 1:

If the Accumulator contains 63H, the HL register pair contains 6000H and memory location 6000H contains 60H, the instruction

CP (HL)

will result in all the flags being reset except N.

Example: 2

If the Accumulator contains 65H and register C also contains 65H, then after the execution of

CP C

the Z flag will be set.

See Appendix E for more details of condition codes affected.

INC r

INCrement

Operation: $r \Leftarrow r + 1$

Format:

Mnemonic: INC **Operands:** r

Object Code:

0	0	r	r	r	1	0	0

Description:

Register r is incremented. r identifies any of the registers A, B, C, D, E, H or L, assembled as follows in the object code.

Register		r
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if r was 7FH before operation; reset otherwise
N:	Reset
C:	Not affected

Example:

If the contents of register D are 28H, after the execution of

INC D

the contents of register D will be 29H.

INC (HL)

INCrement

Operation: (HL) ⇦ (HL) + 1

Format:

Mnemonic: INC **Operands:** (HL)

Object Code:

0	0	1	1	0	1	0	0

34

Description:

The byte contained in the address specified by the contents of the HL register pair is incremented.

M cycles: 3 T states: 11(4,4,3) 4 MHz E.T.: 2.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if (HL) was 7FH before operation; reset otherwise
N: Reset
C: Not Affected

Example:

If the contents of the HL register pair are 3434H, and the contents of address 3434H are 82H, after the execution of

INC (HL)

memory location 3434H will contain 83H.

INC (IX+d)

INCrement

Operation: (IX+d) ⇦ (IX+d) + 1

Format:

Mnemonic: INC **Operands:** (IX+d)

Object Code:

1	1	0	1	1	1	0	1	DD
0	0	1	1	0	1	0	0	34
d	d	d	d	d	d	d	d	

Description:

The contents of the Index Register IX (register pair IX) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M cycles: 6 T states: 23(4,4,3,5,4,3) 4 MHz E.T.: 5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if (IX+d) was 7FH before operation; reset otherwise
N: Reset
C: Not affected

Example:

If the contents of the Index Register pair IX are 2020H, and the memory location 2030H contains byte 34H, after the execution of

INC (IX+10H)

the contents of memory location 2030H will be 35H.

INC (IY + d)

INCrement

Operation: (IY+d) ⇦ (IY+d) + 1

Format:

Mnemonic: INC **Operands:** (IY+d)

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	0	0	34
d	d	d	d	d	d	d	d	

Description:

The contents of the Index Register IY (register pair IY) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M cycles: 6 T states: 23(4,4,3,5,4,3) 4 MHz E.T.: 5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if (IY+d) was 7FH before operation; reset otherwise
N: Reset
C: Not Affected

Example:

If the contents of the Index Register pair IY are 2020H, and the memory location 2030H contain byte 34H, after the execution of

INC (IY+10H)

the contents of memory location 2030H will be 35H.

DEC m

DECrement

Operation: m ⇦ m − 1

Format:

Mnemonic: DEC **Operands:** m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous INC instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

DEC r	0	0	r	r	r	1	0	1	
DEC (HL)	0	0	1	1	0	1	0	1	35
DEC (IX+d)	1	1	0	1	1	1	0	1	DD
	0	0	1	1	0	1	0	1	35
	d	d	d	d	d	d	d	d	
DEC (IY+d)	1	1	1	1	1	1	0	1	FD
	0	0	1	1	0	1	0	1	35
	d	d	d	d	d	d	d	d	

r identifies register A, B, C, D, E, H or L assembled as follows in the object code field above:

Register		**r**
A	=	111
B	=	000
C	=	001
D	=	010
E	=	011
H	=	100
L	=	101

Description:

The byte specified by the m operand is decremented.

Instruction	M Cycles	T States	4 MHz E.T. in μs
DEC r	1	4	1.00
DEC (HL)	3	11(4,4,3)	2.75
DEC (IX+d)	6	23(4,4,3,5,4,3)	5.75
DEC (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Set if m was 80H before operation; reset otherwise
N: Set
C: Not affected

Example:

If the D register contains byte 2AH, after the execution of

DEC D

register D will contain 29H.

General Purpose Arithmetic and CPU Control Groups

DAA

Operation: Decimal-Adjust Accumulator

Format:

Mnemonic: DAA **Operands:**

Object Code:

0	0	1	0	0	1	1	1

Description:

This instruction modifies the results of addition or subtraction so that the results of binary arithmetic are correct for decimal numbers. The Binary Coded Decimal (BCD) code uses the 8-bit accumulator as follows: the eight bits are broken up into two groups of four bits, which represent a two-digit decimal number from 00 to 99. If numbers like this are added with the binary adder in the Z-80, answers larger than 10 may result in each decimal place. The DAA instruction will "adjust" the answer so that each decimal place has a value of 9 or less, and so that the digits have the correct decimal value, though they were added by a binary circuit. The carry and half-carry flags are used in this conversion, as is a circuit that detects digits that are 10 or bigger.

Operation	C Before DAA	HEX Value in Upper Digit (bits 7-4)	H Before DAA	HEX Value in Lower Digit (bits 3-0)	Number Added to Byte	C After DAA
	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
ADD	0	A-F	0	0-9	60	1
ADC	0	9-F	0	A-F	66	1
INC	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB	0	0-9	0	0-9	00	0
SBC	0	0-8	1	6-F	FA	0
DEC	1	7-F	0	0-9	A0	1
NEG	1	6-F	1	6-F	9A	1

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S:	Set if most significant bit of Acc. is 1 after operation; reset otherwise
Z:	Set if Acc. is zero after operation; reset otherwise
H:	See instruction
P/V:	Set if Acc. is even parity after operation; reset otherwise
N:	Not affected
C:	See instruction

Example:

If an addition operation is performed between 15 (BCD) and 27 (BCD), simple decimal arithmetic gives this result:

```
  15
+ 27
----
  42
```

But when the binary representations are added in the Accumulator according to standard binary arithmetic,

```
  0001 0101
+ 0010 0111
-----------
  0011 1100 = 3C
```

the sum is not decimal. The DAA instruction adjusts this result so that the correct BCD representation is obtained:

```
  0011 1100
+ 0000 0110 (adding 06 from table)
-----------
  0100 0010 = 42
```

CPL

ComPLement

Operation: $A \Leftarrow \overline{A}$

Format:

Mnemonic: CPL **Operands:**

Object Code:

0	0	1	0	1	1	1	1

2F

Description:

Contents of the Accumulator (register A) are inverted (one's complement).

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set
P/V: Not affected
N: Set
C: Not affected

Example:

If the contents of the Accumulator are 1011 0100, after the execution of

CPL

the Accumulator contents will be 0100 1011.

NEG

NEGate

Operation: A ◊ 0 − A

Format:

Mnemonic: NEG **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	0	44

Description:

Contents of the Accumulator are negated (two's complement). This is the same as subtracting the contents of the Accumulator from zero. Note that 80H is left unchanged.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Set if Acc. was 80H before operation; reset otherwise
N: Set
C: Set if Acc. was not 00H before operation; reset otherwise

Example:

If the contents of the Accumulator are

1	0	0	1	1	0	0	0

after the execution of

NEG

the Accumulator contents will be

0	1	1	0	1	0	0	0

CCF

Complement Carry Flag

Operation: $CY \Leftarrow \overline{CY}$

Format:

Mnemonic: CCF **Operands:**

Object Code:

0	0	1	1	1	1	1	1	3F

Description:

The C flag in the F register is inverted.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Previous carry will be copied
P/V: Not affected
N: Reset
C: Set if CY was 0 before operation; reset otherwise

SCF

Set Carry Flag

Operation: CY◊1

Format:

Mnemonic: SCF **Operands:**

Object Code:

0	0	1	1	0	1	1	1

37

Description:

The C flag in the F register is set.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Set

NOP

No OPeration

Operation:

Format:

Mnemonic: NOP **Operands:**

Object Code:

0	0	0	0	0	0	0	0

00

Description:

CPU performs no operation during this machine cycle.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

HALT

Operation:

Format:

Mnemonic: HALT **Operands:**

Object Code:

0	1	1	1	0	1	1	0

76

Description:

The HALT instruction suspends CPU operation until a subsequent interrupt or reset is received. While in the halt state, the processor will execute NOP's to maintain memory refresh logic.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

DI

Disable Interrupts

Operation: IFF ◊ 0

Format:

Mnemonic: DI **Operands:**

Object Code:

1	1	1	1	0	0	1	1

F3

Description:

DI disables the maskable interrupt by resetting the interrupt enable flip-flops (IFF1 and IFF2). Note that this instruction disables the maskable interrupt during its execution.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

Example:

When the CPU executes the instruction

DI

the maskable interrupt is disabled until it is subsequently re-enabled by an EI instruction. The CPU will not respond to an Interrupt Request (INT) signal.

EI

Enable Interrupts

Operation: IFF ◊ 1

Format:

Mnemonic: EI **Operands:**

Object Code:

1	1	1	1	1	0	1	1

FB

Description:

EI enables the maskable interrupt by setting the interrupt enable flip-flops (IFF1 and IFF2). Note that this instruction disables the maskable interrupt during its execution.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

Example:

When the CPU executes instruction

RETI

the maskable interrupt is enabled. The CPU will now respond to an Interrupt Request (INT) signal.

IM Ø

Interrupt Mode Ø

Operation:

Format:

Mnemonic: IM **Operands:** Ø

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	1	0	46

Description:

The IM Ø instruction sets interrupt mode Ø. In this mode the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. The first byte of a multi-byte instruction is read during interrupt acknowledge cycle. Subsequent bytes are read in by a normal memory read sequence.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.ØØ

Condition Bits Affected: None

IM 1

Interrupt Mode 1

Operation:

Format:

Mnemonic: IM **Operands:** 1

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	1	0	1	1	0	56

Description:

The IM instruction sets interrupt mode 1. In this mode the processor will respond to an interrupt by executing a restart to location 0038H.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected: None

IM 2

Interrupt Mode 2

Operation:

Format:

Mnemonic: IM **Operands:** 2

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	1	1	1	1	0	5E

Description:

The IM 2 instruction sets interrupt mode 2. This mode allows an indirect call to any location in memory. With this mode the CPU forms a 16-bit memory address. The upper eight bits are the contents of the Interrupt Vector Register I and the lower eight bits are supplied by the interrupting device.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected: None

16 Bit Arithmetic Group

ADD HL,ss

Operation: HL ⇦ HL + ss

Format:

Mnemonic: ADD **Operands:** HL, ss

Object Code:

0	0	s	s	1	0	0	1

Description:

The contents of register pair ss (any of register pairs BC, DE, HL or SP) are added to the contents of register pair HL, and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M cycles: 3 T states: 11(4,4,3) 4 MHz E.T.: 2.75

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set if carry out of Bit 11; reset otherwise
P/V: Not affected
N: Reset
C: Set if carry from Bit 15; reset otherwise

Example:

If register pair HL contains the integer 4242H and register pair DE contains 1111H, after the execution of

ADD HL, DE

the HL register pair will contain 5353H.

ADC HL,ss

ADd with Carry

Operation: HL⇦HL+ss+CY

Format:

Mnemonic: ADC **Operands:** HL, ss

Object Code:

1	1	1	0	1	1	0	1

ED

0	1	s	s	1	0	1	0

Description:

The contents of register pair ss (any of register pairs BC, DE, HL or SP) are added with the Carry Flag (C flag in the F register) to the contents of register pair HL, and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M cycles: 4 T states: 15(4,4,4,3) 4 MHz E.T.: 3.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry out of Bit 11; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 15; reset otherwise

Example:

If the register pair BC contains 2222H, register pair HL contains 5437H and the Carry Flag is set, after the execution of

```
ADC  HL,  BC
```

the contents of HL will be 765AH.

SBC HL,ss

SuBtract with Carry

Operation: HL ◊ HL − ss − CY

Format:

Mnemonic: SBC **Operands:** HL, ss

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	s	s	0	0	1	0	

Description:

The contents of the register pair ss (any of register pairs BC, DE, HL or SP) and the Carry Flag (C flag in the F register) are subtracted from the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M cycles: 4 T states: 15(4,4,4,3) 4 MHz E.T.: 3.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from Bit 12; reset otherwise
P/V: Set if overflow; reset otherwise
N: Set
C: Set if borrow; reset otherwise

Example:

If the contents of the HL register pair are 9999H, the contents of register pair DE are 1111H, and the Carry Flag is set, after the execution of

SBC HL, DE

the contents of HL will be 8887H.

ADD IX,pp

Operation: IX◊IX+pp

Format:

Mnemonic: ADD **Operands:** IX,pp

Object Code:

1	1	0	1	1	1	0	1

DD

0	0	p	p	1	0	0	1

Description:

The contents of register pair pp (any of register pairs BC, DE, IX or SP) are added to the contents of the Index Register IX, and the results are stored in IX. Operand pp is specified as follows in the assembled object code.

Register Pair	pp
BC	ØØ
DE	Ø1
IX	1Ø
SP	11

M cycles: 4 T states: 15(4,4,4,3) 4 MHz E.T.: 3.75

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set if carry out of Bit 11; reset otherwise
P/V: Not affected
N: Reset
C: Set if carry from Bit 15; reset otherwise

Example:

If the contents of Index Register IX are 3333H and the contents of register pair BC are 5555H, after the execution of

ADD IX, BC

the contents of IX will be 8888H.

ADD IY,rr

Operation: IY ⇐ IY + rr

Format:

Mnemonic: ADD **Operands:** IY, rr

Object Code:

1	1	1	1	1	1	0	1

FD

0	0	r	r	1	0	0	1

Description:

The contents of register pair rr (any of register pairs BC, DE, IY or SP) are added to the contents of Index Register IY, and the result is stored in IY. Operand rr is specified as follows in the assembled object code.

Register Pair	rr
BC	00
DE	01
IY	10
SP	11

M cycles: 4 T states: 15(4,4,4,3) 4 MHz E.T.: 3.75

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set if carry out of Bit 11; reset otherwise
P/V: Not affected
N: Reset
C: Set if carry from Bit 15; reset otherwise

Example:

If the contents of Index Register IY are 333H and the contents of register pair BC are 555H, after the execution of

ADD IY, BC

the contents of IY will be 888H.

INC ss

INCrement

Operation: ss ◊ ss + 1

Format:

Mnemonic: INC **Operands:** ss

Object Code:

0	0	s	s	0	0	1	1

Description:

The contents of register pair ss (any of register pairs BC, DE, HL or SP) are incremented. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M cycles: 1 T states: 6 4 MHz E.T.: 1.50

Condition Bits Affected: None

Example:

If the register pair contains 1000H, after the execution of

INC HL

HL will contain 1001H.

INC IX

INCrement

Operation: IX ◊ IX + 1

Format:

Mnemonic: INC **Operands:** IX

Object Code:

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	1	1	23

Description:

The contents of the Index Register IX are incremented.

M cycles: 2 T states: 10(4,6) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the Index Register IX contains the integer 3300H after the execution of
INC IX
the contents of Index Register IX will be 3301H.

INC IY

INCrement

Operation: IY ◊ IY + 1

Format:

Mnemonic: INC **Operands:** IY

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	1	1	23

Description:

The contents of the Index Register IY are incremented.

M cycles: 2 T states: 10(4,6) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Index Register are 2977H, after the execution of

INC IY

the contents of Index Register IY will be 2978H.

DEC ss

DECrement

Operation: ss ⇦ ss − 1

Format:

Mnemonic: DEC **Operands:** ss

Object Code:

0	0	s	s	1	0	1	1

Description:

The contents of register pair ss (any of the register pairs BC, DE, HL or SP) are decremented. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M cycles: 1 T states: 6 4 MHz E.T.: 1.50

Condition Bits Affected: None

Example:

If register pair HL contains 1001H, after the execution of

DEC HL

the contents of HL will be 1000H.

DEC IX

DECrement

Operation: IX ⇦ IX − 1

Format:

Mnemonic: DEC **Operands:** IX

Object Code:

1	1	0	1	1	1	0	1	DD
0	0	1	0	1	0	1	1	2B

Description:

The contents of Index Register IX are decremented.

M cycles: 2 T states: 10(4,6) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of Index Register IX are 2006H, after the execution of

DEC IX

the contents of Index Register IX will be 2005H.

DEC IY

DECrement

Operation: IY ⇦ IY − 1

Format:

Mnemonic: DEC **Operands:** IY

Object Code:

1	1	1	1	1	1	0	1	FD
0	0	1	0	1	0	1	1	2B

Description:

The contents of the Index Register IY are decremented.

M cycles: 2 T states: 10(4,6) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Index Register IY are 7649H, after the execution of

DEC IY

the contents of Index Register IY will be 7648H.

Rotate and Shift Group

RLCA — Rotate Left Circular Accumulator

Operation: CY ← 7 ← 0 ← (A)

Format:

Mnemonic: RLCA **Operands:**

Object Code:

0	0	0	0	0	1	1	1

07

Description:
The contents of the Accumulator (register A) are rotated left: the content of bit 0 is moved to bit 1; the previous content of bit 1 is moved to bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. (Bit 0 is the least significant bit.)

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from Bit 7 of Acc.

Example:

If the contents of the Accumulator are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

RLCA

the contents of the Carry Flag and the Accumulator will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLA

Rotate Left Accumulator

Operation: CY ← 7 ← 0 ← (A)

Format:

Mnemonic: RLA **Operands:**

Object Code:

0	0	0	1	0	1	1	1

17

Description:

The contents of the Accumulator (register A) are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 0. Bit 0 is the least significant bit.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from Bit 7 of Acc.

Example:

If the contents of the Carry Flag and the Accumulator are

C	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	0

after the execution of

RLA

the contents of the Carry Flag and the Accumulator will be

C	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	1

RRCA

Rotate Right Circular Accumulator

Operation: 7→0 (A) → CY

Format:

Mnemonic: RRCA **Operands:**

Object Code:

0	0	0	0	1	1	1	1

0F

Description:

The contents of the Accumulator (register A) are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the register. The content of bit 0 is copied into bit 7 and also into the Carry Flag (C flag in register F.) Bit 0 is the least significant bit.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from Bit 0 of Acc.

Example:

If the contents of the Accumulator are

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1

After the execution of

RRCA

the contents of the Accumulator and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	0	0	0	1	0	0	0	1

RRA

Rotate Right Accumulator

Operation: 7→0 → CY (A)

Format:

Mnemonic: RRA **Operands:**

Object Code:

0	0	0	1	1	1	1	1

1F

Description:

The contents of the Accumulator (register A) are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the register. The content of bit 0 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 7. Bit 0 is the least significant bit.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from Bit 0 of Acc.

Example:

If the contents of the Accumulator and the Carry Flag are

7	6	5	4	3	2	1	0	C
1	1	1	0	0	0	0	1	0

after the execution of

RRA

the contents of the Accumulator and the Carry Flag will be

7	6	5	4	3	2	1	0	C
0	1	1	1	0	0	0	0	1

RLC r

Rotate Left Circular

Operation: CY ← 7 ← 0 ← (r)

Format:

Mnemonic: RLC **Operands:** r

Object Code:

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	r	r	r	

Description:

The eight-bit contents of register r are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Operand r is specified as follows in the assembled object code:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Note: Bit 0 is the least significant bit.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from Bit 7 of source register

Example:

If the contents of register r are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

RLC r

the contents of the Carry Flag and register r will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLC (HL)

Rotate Left Circular

Operation: CY ⇦ 7⇦0 ⇦ (HL)

Format:

Mnemonic: RLC **Operands:** (HL)

Object Code:

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	1	1	0	Ø6

Description:

The contents of the memory address specified by the contents of register pair HL are rotated left: the content of bit Ø is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit Ø. Bit Ø is the least significant bit.

M cycles: 4 T states: 15(4,4,4,3) 4 MHz E.T.: 3.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from Bit 7 of source register

Example:

If the contents of the HL register pair are 2828H, and the contents of memory location 2828H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

RLC (HL)

the contents of the Carry Flag and memory locations 2828H will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLC (IX+d)

Rotate Left Circular

Operation: CY ⇦ 7⇦0 ⇦ (IX+d)

Format:

Mnemonic: RLC **Operands:** (IX+d)

Object Code:

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
0	0	0	0	0	1	1	0	Ø6

Description:

The contents of the memory address specified by the sum of the contents of the Index Register IX and a two's complement displacement integer d, are rotated left: the contents of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

M cycles: 6 T states: 23(4,4,3,5,4,3) 4 MHz E.T.: 5.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of the Index Register IX are 1000H, and the contents of memory location 1002H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

RLC (IX+2H)

the contents of the Carry Flag and memory location 1002H will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLC (IY+d)

Rotate Left Circular

Operation: CY ← 7 ← 0 ← (IY+d)

Format:

Mnemonic: RLC **Operands:** (IY+d)

Object Code:

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
0	0	0	0	0	1	1	0	06

Description:

The contents of the memory address specified by the sum of the contents of the Index Register IY and a two's complement displacement integer d are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this process is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

M cycles: 6 T states: 23(4,4,3,5,4,3) 4 MHz E.T.: 5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from Bit 7 of source register

Example:

If the contents of the Index Register IY are 1000H, and the contents of memory location 1002H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

RLC (IY+2H)

the contents of the Carry Flag and memory location 1002H will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RL m

Rotate Left

Operation: CY ⇦ 7⇦0 ⇦ (m)

Format:

Mnemonic: RL **Operands:** m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

Object Code:

RL r	1	1	0	0	1	0	1	1	CB
	0	0	0	1	0	r	r	r	
RL (HL)	1	1	0	0	1	0	1	1	CB
	0	0	0	1	0	1	1	0	16
RL (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	0	1	0	1	1	0	16

RL (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	0	1	0	1	1	0	16

r identifies register B, C, D, E, H, L or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of the m operand are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 0. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T. in μs
RL r	2	8(4,4)	2.00
RL (HL)	4	15(4,4,4,3)	3.75
RL (IX+d)	6	23(4,4,3,5,4,3)	5.75
RL (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of the Carry Flag and register D are

C	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1

after the execution of

RL D

the contents of the Carry Flag and register D will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	0

RRC m

Rotate Right Circular

Operation: ⇨ [7 ⇨ 0] ⇨ [CY]
m

Format:

Mnemonic: RRC **Operands:** m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

Object Code:

RRC r	1	1	0	0	1	0	1	1	CB
	0	0	0	0	1	r	r	r	
RRC (HL)	1	1	0	0	1	0	1	1	CB
	0	0	0	0	1	1	1	0	0E

RRC (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	0	0	1	1	1	0	ØE
RRC (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	0	0	1	1	1	0	ØE

r identifies register B, C, D, E, H, L or A specified as follows in the assembled object code above:

Register	r
B	ØØØ
C	ØØ1
D	Ø1Ø
E	Ø11
H	1ØØ
L	1Ø1
A	111

Description:

The contents of operand m are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit Ø is copied into the Carry Flag (C flag in the F register) and also into bit 7. Bit Ø is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T. in μs
RRC r	2	8(4,4)	2.00
RRC (HL)	4	15(4,4,4,3)	3.75
RRC (IX+d)	6	23(4,4,3,5,4,3)	5.75
RRC (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from Bit 0 of source register

Example:

If the contents of register A are

7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	1

after the execution of

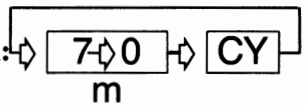
RRC A

the contents of register A and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	0	0	1	1	0	0	0	1

RR m

Rotate Right

Operation: 

Format:

Mnemonic: RR **Operands:** m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

Object Code:

RR r	1	1	0	0	1	0	1	1	CB
	0	0	0	1	1	r	r	r	
RR (HL)	1	1	0	0	1	0	1	1	CB
	0	0	0	1	1	1	1	0	1E
RR (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	0	1	1	1	1	0	1E
RR (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	0	1	1	1	1	0	1E

r identifies registers B, C, D, E, H, L or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are rotated right: the contents of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 7. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T. in μs
RR r	2	8(4,4)	2.00
RR (HL)	4	15(4,4,4,3)	3.75
RR (IX+d)	6	23(4,4,3,5,4,3)	5.75
RR (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Data from Bit 0 of source register

Example:

If the contents of the HL register pair are 4343H, and the contents of memory location 4343H and the Carry Flag are

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	1	0

after the execution of

RR (HL)

the contents of location 4343H and the Carry Flag will be

7	6	5	4	3	2	1	0	C
0	1	1	0	1	1	1	0	1

SLA m

Shift Left Arithmetic

Operation: CY ← 7 ← 0 ← 0
m

Format:

Mnemonic: SLA **Operands:** m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

Object Code:

SLA r	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	r	r	r	
SLA (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	1	1	0	26
SLA (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	1	0	0	1	1	0	26
SLA (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	1	0	0	1	1	0	26

r identifies registers B, C, D, E, H, L or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

An arithmetic shift left is performed on the contents of operand m: bit 0 is reset, the previous content of bit 0 is copied into bit 1, the previous content of bit 1 is copied into bit 2; this pattern is continued throughout; the content of bit 7 is copied into the Carry Flag (C flag in register F). Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T. in μs
SLA r	2	8(4,4)	2.00
SLA (HL)	4	15(4,4,4,3)	3.75
SLA (IX+d)	6	23(4,4,3,5,4,3)	5.75
SLA (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 7

Example:

If the contents of register L are

7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1

after the execution of

SLA L

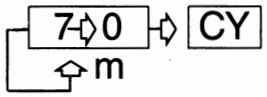
the contents of the Carry Flag and register L will be

C	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1	0

SRA m

Shift Right Arithmetic

Operation:



Format:

Mnemonic: SRA **Operands:** m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

Object Code:

SRA r	1	1	0	0	1	0	1	1	CB
	0	0	1	0	1	r	r	r	
SRA (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	0	1	1	1	0	2E
SRA (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	1	0	1	1	1	0	2E

SRA (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	1	0	1	1	1	0	2E

r means register B, C, D, E, H, L or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

An arithmetic shift right is performed on the contents of operand m: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in register F), and the previous content of bit 7 is unchanged. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T. in μs
SRA r	2	8(4,4)	2.00
SRA (HL)	4	15(4,4,4,3)	3.75
SRA (IX+d)	6	23(4,4,3,5,4,3)	5.75
SRA (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of the Index Register IX are 1000H, and the contents of memory location 1003H are

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

after the execution of

SRA (IX+3H)

the contents of memory location 1003H and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	0	0

SRL m

Shift Right Logical

Operation: 0 ⇨ [7 ⇨ 0] ⇨ [CY]
m

Format:

Mnemonic: SRL **Operands:** m

The operand m is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

Object Code:

SRL r	1	1	0	0	1	0	1	1	CB
	0	0	1	1	1	r	r	r	
SRL (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	1	1	1	1	0	3E

SRL (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	1	1	1	1	1	0	3E
SRL (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	0	0	1	1	1	1	1	0	3E

r identifies registers B, C, D, E, H, L or A specified as follows in the assembled object code fields above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are shifted right: the content of bit 7 is copied into bit 6; the content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag, and bit 7 is reset. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T. in μs
SRL r	2	8(4,4,)	2.00
SRL (HL)	4	15(4,4,4,3)	3.75
SRL (IX+d)	6	23(4,4,3,5,4,3)	5.75
SRL (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Data from Bit 0 of source register

Example:

If the contents of register B are

7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1

after the execution of

SRL B

the contents of register B and the Carry Flag will be

7	6	5	4	3	2	1	0	C
0	1	0	0	0	1	1	1	1

RLD

Rotate Left Decimal

Operation: A 7 4 3 0 7 4 3 0 (HL)

Format:

Mnemonic: RLD **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED

0	1	1	0	1	1	1	1	6F

Description:

The contents of the low order four bits (bits 3, 2, 1 and Ø) of the memory location (HL) are copied into the high order four bits (7, 6, 5 and 4) of that same memory location; the previous contents of those high order four bits are copied into the low order four bits of the Accumulator (register A), and the previous contents of the low order four bits of the Accumulator are copied into the low order four bits of memory location (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

M cycles: 5 T states: 18(4,4,3,4,3) 4 MHz E.T.: 4.5Ø

Condition Bits Affected:

S: Set if Acc. is negative after operation; reset otherwise
Z: Set if Acc. is zero after operation; reset otherwise
H: Reset
P/V: Set if parity of Acc. is even after operation; reset otherwise
N: Reset
C: Not affected

Example:

If the contents of the HL register pair are 5ØØØH, and the contents of the Accumulator and memory location 5ØØØH are

7	6	5	4	3	2	1	0	
0	1	1	1	1	0	1	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	1	0	0	0	1	(5ØØØH)

after the execution of

RLD

the contents of the Accumulator and memory location 5000H will be

7	6	5	4	3	2	1	0	
0	1	1	1	0	0	1	1	Accumulator

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	0	(5000H)

RRD

Rotate Right Decimal

Operation: A 7430 7430 (HL)

Format:

Mnemonic: RRD **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	1	0	0	1	1	1	67

Description:

The contents of the low order four bits (bits 3, 2, 1 and 0) of memory location (HL) are copied into the low order four bits of the Accumulator (register A); the previous contents of the low order four bits of the Accumulator are copied into the high order four bits (7, 6, 5 and 4) of location (HL); and the previous contents of the high order four bits of (HL) are copied into the low order four bits of (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

M cycles: 5 T states: 18(4,4,3,4,3) 4 MHz E.T.: 4.50

Condition Bits Affected:

S:	Set if Acc. is negative after operation; reset otherwise
Z:	Set if Acc. is zero after operation; reset otherwise
H:	Reset
P/V:	Set if parity of Acc. is even after operation; reset otherwise
N:	Reset
C:	Not affected

Example:

If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are

7	6	5	4	3	2	1	0	
1	0	0	0	0	1	0	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	(5000H)

after the execution of

RRD

the contents of the Accumulator and memory location 5000H will be

7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	Accumulator

7	6	5	4	3	2	1	0	
0	1	0	0	0	0	1	0	(5000H)

Bit Set, Reset and Test Group

BIT b, r

BIT test

Operation: $Z \Leftarrow \overline{r_b}$

Format:

Mnemonic: BIT **Operands:** b, r

Object Code:

1	1	0	0	1	0	1	1

CB

0	1	b	b	b	r	r	r

Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the indicated register. Operands b and r are specified as follows in the assembled object code:

Bit Tested	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected:

S: Unknown
Z: Set if specified Bit is 0; reset otherwise
H: Set
P/V: Unknown
N: Reset
C: Not affected

Example:

If bit 2 in register B contains 0, after the execution of

BIT 2, B

the Z flag in the F register will contain 1, and bit 2 in register B will remain 0. (Bit 0 in register B is the least significant bit.)

BIT b,(HL)

Bit Test

Operation: $Z \Leftarrow \overline{(HL)_b}$

Format:

Mnemonic: BIT **Operands:** b, (HL)

Object Code:

1	1	0	0	1	0	1	1	CB
0	1	b	b	b	1	1	0	

Description:

This instruction tests bit b in the memory location specified by the contents of the HL register pair and sets the Z flag accordingly. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M cycles: 3 T states: 12(4,4,4) 4 MHz E.T.: 3.00

Condition Bits Affected:

S:	Unknown
Z:	Set if specified Bit is Ø; reset otherwise
H:	Set
P/V:	Unknown
H:	Reset
C:	Not affected

Example:

If the HL register pair contains 444H, and bit 4 in the memory location 444H contains 1, after the execution of

BIT 4,(HL)

the Z flag in the F register will contain Ø, and bit 4 in memory location 444H will still contain 1. (Bit Ø in memory location 444H is the least significant bit.)

BIT b,(IX+d)

Bit Test

Operation: $Z \Leftarrow \overline{(IX+d)_b}$

Format:

Mnemonic: BIT **Operands:** b, (IX+d)

Object Code:

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
0	1	b	b	b	1	1	0	

Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents register pair IX (Index Register IX) and the two's complement displacement integer d. Operand b is specified as follows in the assembled object code.

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M cycles: 5 T states: 20(4,4,3,5,4) 4 MHz E.T.: 5.00

Condition Bits Affected:

S: Unknown
Z: Set if specified Bit is 0; reset otherwise
H: Set
P/V: Unknown
N: Reset
C: Not affected

Example:

If the contents of Index Register IX are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of

BIT 6,(IX+4H)

the Z flag in the F register will contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

BIT b,(IY+d)

BIT Test

Operation: $Z \Diamond \overline{(IY+d)_b}$

Format:

Mnemonic: BIT **Operands:** b, (IY+d)

Object Code:

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
0	1	b	b	b	1	1	0	

Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents of register pair IY (Index Register IY) and the two's complement displacement integer d. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M cycles: 5 T states: 20(4,4,3,5,4) 4 MHz E.T.: 5.00

Condition Bits Affected:

S: Unknown
Z: Set if specified Bit is 0; reset otherwise
H: Set
P/V: Unknown
N: Reset
C: Not affected

Example:

If the contents of Index Register are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of

BIT 6,(IY+4H)

the Z flag in the F register still contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

SET b,r

Operation: $r_b \Leftarrow 1$

Format:

Mnemonic: SET **Operands:** b, r

Object Code:

1	1	0	0	1	0	1	1

CB

1	1	b	b	b	r	r	r

Description:

Bit b (any bit, 7 through 0) in register r (any of register B, C, D, E, H, L or A) is set. Operands b and r are specified as follows in the assembled object code:

Bit Tested	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected: None

Example:

After the execution of

SET 4,A

bit 4 in register A will be set. (Bit 0 is the least significant bit.)

SET b,(HL)

Operation: $(HL)_b \Leftarrow 1$

Format:

Mnemonic: SET **Operands:** b, (HL)

Object Code:

1	1	0	0	1	0	1	1	CB

1	1	b	b	b	1	1	0

Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the contents of register pair HL is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M cycles: 4 T states: 15(4,4,4,3) 4 MHz E.T.: 3.75

Condition Bits Affected: None

Example:

If the contents of the HL register pair are 3000H, after the execution of

SET 4,(HL)

bit 4 in memory location 3000H will be 1. (Bit 0 in memory location 3000H is the least significant bit.)

SET b,(IX+d)

Operation: $(IX+d)_b \Leftarrow 1$

Format:

Mnemonic: SET **Operands:** b, (IX+d)

Object Code:

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
1	1	b	b	b	1	1	0	

Description:

Bit b (any bit, 7 through Ø) in the memory location addressed by the sum of the contents of the IX register pair (Index Register IX) and the two's complement integer d is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
Ø	ØØØ
1	ØØ1
2	Ø1Ø
3	Ø11
4	1ØØ
5	1Ø1
6	11Ø
7	111

M cycles: 6 T states: 23(4,4,3,5,4,3) 4 MHz E.T.: 5.75

Condition Bits Affected: None

Example:

If the contents of Index Register are 2ØØØH, after the execution of

SET Ø,(IX+3H)

bit Ø in memory location 2ØØ3H will be 1. (Bit Ø in memory location 2ØØ3H is the least significant bit.)

SET b,(IY+d)

Operation: $(IY+d)_b \Leftarrow 1$

Format:

Mnemonic: SET **Operands:** b, (IY+d)

Object Code:

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
1	1	b	b	b	1	1	0	

Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the sum of the contents of the IY register pair (Index Register IY) and the two's complement displacement d is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M cycles: 6 T states: 23(4,4,3,5,4,3) 4 MHz E.T.: 5.75

Condition Bits Affected: None

Example:

If the contents of Index Register IY are 2000H, after the execution of

SET 0,(IY+3H)

bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

RES b,m

RESet

Operation: $S_b \Leftarrow 0$

Format:

Mnemonic: RES **Operands:** b, m

Operand b is any bit (7 through 0) of the contents of the m operand, (any of r, (HL), (IX+d) or (IY+d) as defined for the analogous SET instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

Object Code:

RES b, r	1	1	0	0	1	0	1	1	CB
	1	0	b	b	b	r	r	r	
RES b, (HL)	1	1	0	0	1	0	1	1	CB
	1	0	b	b	b	1	1	0	
RES b, (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	d	d	d	d	d	d	d	d	
	1	0	b	b	b	1	1	0	

RES b, (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d	d	d	d	d	d	d	d	
1	0	b	b	b	1	1	0	

Bit Reset	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

Description:

Bit b in operand m is reset.

Instruction	M Cycles	T States	4 MHz E.T. in μs
RES r	4	8(4,4)	2.00
RES (HL)	4	15(4,4,4,3)	3.75
RES (IX+d)	6	23(4,4,3,5,4,3)	5.75
RES (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected: None

Example 1:

After the execution of

RES 6,D (object code CB, B2H)

bit 6 in register D will be reset. (Bit 0 in register D is the least significant bit.)

Example 2:

If HL contains 7000H and address 7000H contains FFH, after

RES 0,(HL)

address 7000H will contain FEH.

Jump Group

JP nn

JumP

Operation: PC ⇐ nn

Format:

Mnemonic: JP **Operands:** nn

Object Code:

1	1	0	0	0	0	1	1

C3

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Note: The first operand in this assembled object code is the low order byte of a 2-byte address.

Description:

Operand nn is loaded into register pair PC (Program Counter) and points to the address of the next program instruction to be executed.

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

JP 50A1

This instruction will cause the program to jump to the instruction at 50A1H by loading the number 50A1H into the PC register.

JP cc,nn

JumP

Operation: IF cc TRUE, PC◊nn

Format:

Mnemonic: JP **Operands:** cc, nn

Object Code:

1	1	cc	cc	cc	0	1	0

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Note: The first n operand in this assembled object code is the low order byte of a 2-byte memory address.

Description:

If condition cc is true, the instruction loads operand nn into register pair PC (Program Counter), and the program continues with the instruction beginning at address nn. If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status bits which correspond to condition bits in the Flag Register (register F). These eight status bits are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code.

cc	Condition	Relevant Flag
000	NZ non-zero	Z (=0)
001	Z zero	Z (=1)
010	NC no-carry	C (=0)
011	C carry	C (=1)
100	PO parity odd	P/V (=0)
101	PE parity even	P/V (=1)
110	P sign positive	S (=0)
111	M sign negative	S (=1)

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the Carry Flag (C flag in the F register) is set and the contents of address 1520 are 03H, after the execution of

JP C,1520H

the Program Counter will contain 1520H, and on the next machine cycle the CPU will fetch from address 1520H the byte 03H.

JR e

Jump Relative

Operation: PC ⇐ PC + e

Format:

Mnemonic: JR **Operands:** e

Object Code:

0	0	0	1	1	0	0	0

18

e-2	e-2	e-2	e-2	e-2	e-2	e-2	e-2

Description:

This instruction provides for unconditional branching to other segments of a program. The value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. This jump as measured from the address of the instruction opcode has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

M cycles: 3 T states: 12(4,3,5) 4 MHz E.T.: 3.00

Condition Bits Affected: None

Example 1:

To jump forward five locations from address 480, the following assembly language statement is used:

JR $+5

The resulting object code and final PC value is shown below:

Location	Instruction
480	18
481	03
482	—◊PC before jump
483	—
484	—
485	◊PC after jump

Note: when using an assembler, $ + 5 used above would normally be replaced by a label.

Example 2:

This program will skip around the NOP instruction.

```
START   JR, END
        NOP
END     —
```

JR C,e

Jump Relative

Operation: If C = 0, continue
If C = 1, PC◊PC + e

Format:

Mnemonic: JR **Operands:** C, e

Object Code:

0	0	1	1	1	0	0	0	38
e-2	e-2	e-2	e-2	e-2	e-2	e-2	e-2	

Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to a '1,' the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump as measured from the address of the instruction opcode has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '0,' the next instruction to be executed is taken from the location following this instruction.

If condition is met:

M cycles: 3 T states: 12(4,3,5) 4 MHz E.T.: 3.00

If condition is not met:

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

The Carry Flag is set and it is required to jump back four locations from 480. The assembly language statement is:

JR C, $-4

The resulting object code and final PC value is shown below:

Location	Instruction
47C	◊PC after jump
47D	—
47E	—
47F	—
480	38
481	FA (two's complement – 6)
482	◊PC before jump

JR NC,e

Jump Relative

Operation: If C = 1, continue
If C = 0, PC◊PC + e

Format:

Mnemonic: JR **Operands:** NC, e

Object Code:

0	0	1	1	0	0	0	0	30
e-2	e-2	e-2	e-2	e-2	e-2	e-2	e-2	

Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to '0,' the value of the displacement e is added to the Program Counter (PC) and

the next instruction is fetched from the location designated by the new contents of the PC. The jump as measured from the address of the instruction opcode has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '1,' the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M cycles: 3 T states: 12(4,3,5) 4 MHz E.T.: 3.00

If the condition is not met:

M cycles: 7 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

The Carry Flag is reset and it is required to repeat the jump instruction. The assembly language statement is:

JR NC,$

The resulting object code and PC after the jump are shown below:

Location	Instruction
480	30 ◊ PC after jump
481	FD (two's complement − 2)
482	— ◊ PC before jump

Note: this instruction would cause an infinite loop in the program.

JR Z,e

Jump Relative

Operation: Z = 0, continue
If Z = 1, PC ◊ PC + e

Format:

Mnemonic: JR **Operands:** Z, e

Object Code:

0	0	1	0	1	0	0	0	28

e-2	e-2	e-2	e-2	e-2	e-2	e-2	e-2

Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '1,' the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump as measured from the address of the instruction opcode has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '0,' the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M cycles: 3 T states: 12(4,3,5) 4 MHz E.T.: 3.00

If the condition is not met:

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

The Zero Flag is set and it is required to jump forward five locations from address 300. The following assembly language statement is used:

JR Z, $+5

The resulting object code and final PC value is shown below:

Location	Instruction
300	28
301	03
302	—◊PC before jump
303	—
304	—
305	—◊PC after jump

JR NZ,e

Jump Relative

Operation: If Z = 1, continue
If Z = 0, PC◊PC + e

Format:

Mnemonic: JR **Operands:** NZ, e

Object Code:

0	0	1	0	0	0	0	0	20

e-2	e-2	e-2	e-2	e-2	e-2	e-2	e-2

Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '0,' the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump as measured from the address of the instruction opcode has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '1,' the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M cycles: 3 T states: 12(4,3,5) 4 MHz E.T.: 3.00

If the condition is not met:

M cycles: 2 T states: 7(4,3) 4 MHz E.T.: 1.75

Condition Bits Affected: None

Example:

The Zero Flag is reset and it is required to jump back four locations from 480. The assembly language statement is:

JR NZ, $−4

The resulting object code and final PC value is shown below:

Location	Instruction
47C	◊PC after jump
47D	—
47E	—
47F	—
480	20
481	FA (two's complement −6)
482	—◊PC before jump

JP (HL)

Jump

Operation: PC ◊ HL

Format:

Mnemonic: JP **Operands:** (HL)

Object Code:

1	1	1	0	1	0	0	1

E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the HL register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M cycles: 1 T states: 4 4 MHz E.T.: 1.00

Condition Bits Affected: None

Example 1:

If the contents of the Program Counter are 1000H and the contents of the HL register pair are 4800H, after the execution of

JP (HL)

the contents of the Program Counter will be 4800H.

The program will jump to the instruction at address 4800H.

Example 2:

A typical software routine which uses JP (HL) is a jump table lookup program. Assume that n 16-bit addresses are listed in consecutive bytes of memory starting at address TBL. Also assume that the Accumulator contains a number from 0 to n-1 representing the routine to be jumped to.

```
LD      HL, TBL  ; HL points to the first byte in the table.
ADD     A, A     ; double A
LD      DE, 0
LD      E, A
ADD     HL, DE   ; if A originally contained 5, then HL now points to the
                   5th address in the table
LD      E, (HL)
INC     HL
LD      D, (HL)  ; DE now contains the 5th address of the table
LD      HL, DE   ; HL now contains the 5th address of the table
JP      (HL)
```

JP (IX)

JumP

Operation: PC ◊ IX

Format:

Mnemonic: JP **Operands:** (IX)

Object Code:

1	1	0	1	1	1	0	1	DD
1	1	1	0	1	0	0	1	E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the IX Register Pair (Index Register IX). The next instruction is fetched from the location designated by the new contents of the PC.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H, and the contents of the IX Register Pair are 4800H, after the execution of

JP (IX)

the contents of the Program Counter will be 4800H.

JP (IY)

JumP

Operation: PC ◊ IY

Format:

Mnemonic: JP **Operands:** (IY)

Object Code:

1	1	1	1	1	1	0	1	FD
1	1	1	0	1	0	0	1	E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the IY Register Pair (Index Register IY). The next instruction is fetched from the location designated by the new contents of the PC.

M cycles: 2 T states: 8(4,4) 4 MHz E.T.: 2.00

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H and the contents of the IY Register Pair are 4800H, after the execution of

JP (IY)

the contents of the Program Counter will be 4800H.

DJNZ e

Decrement Jump Not Zero

Operation:

Format:

Mnemonic: DJNZ **Operands:** e

Object Code:

0	0	0	1	0	0	0	0	10
e-2	e-2	e-2	e-2	e-2	e-2	e-2	e-2	

Description:

The instruction is similar to the conditional jump instructions except that a register value is used to determine branching. The B register is decremented and if a non zero value remains, the value of the displacement e is added to the Program Counter (PC). The next instruction is fetched from the location

designated by the new contents of the PC. The jump is measured from the address of the instruction opcode has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the result of decrementing leaves B with a zero value, the next instruction to be executed is taken from the location following this instruction.

If B ≠ Ø:

M cycles: 3 T states: 13(5,3,5) 4 MHz E.T.: 3.25

If B = Ø:

M cycles: 2 T states: 8(5,3) 4 MHz E.T.: 2.ØØ

Condition Bits Affected: None

Example:

A typical software routine is used to demonstrate the use of the DJNZ instruction. This routine moves a line from an input buffer (INBUF) to an output buffer (OUTBUF). It moves the bytes until it finds a carriage return, or until it has moved 80 bytes, whichever occurs first.

```
        LD      B, 8Ø           ; Set up counter
        LD      HL, Inbuf       ; Set up pointers
        LD      DE, Outbuf
LOOP:   LD      A, (HL)         ; Get next byte from
                                ; input buffer
        LD      (DE), A         ; Store in output buffer
        CP      ØDH             ; Is it a CR?
        JR      Z, DONE         ; Yes finished
        INC     HL              ; Increment pointers
        INC     DE
        DJNZ    LOOP            ; Loop back if 8Ø
                                ; bytes have not
                                ; been moved
DONE:
```

Call and Return Group

CALL nn

Operation: $(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, $PC \leftarrow nn$

Format:

Mnemonic: CALL **Operands:** nn

Object Code:

1	1	0	0	1	1	0	1	CD
n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	

Note: The first of the two n operands in the assembled object code above is the least significant byte of a two-byte memory address.

Description:

After pushing the current contents of the Program Counter (PC) onto the top of the external memory stack, the operands nn are loaded into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into PC.) The push is accomplished by first decrementing the current contents of the Stack Pointer (register pair SP), loading the high-order byte of the PC contents into the memory address now pointed to by the SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of stack. Note: Because this is a three-byte instruction, the Program Counter will have been incremented by three before the push is executed.

M cycles: 5 T states: 17(4,3,4,3,3) 4 MHz E.T.: 4.25

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	CDH
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the three-byte instruction CD3521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL 2135H

After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

Before:

Stack Pointer	Address	Stack
3002	3002	50
	3003	1B
	3004	3C

Program Counter

1A47

After CALL 2135H:

Stack Pointer	Address	Stack
3000	3000	4A
	3001	1A
	3002	50
	3003	1B

Program Counter

2135

CALL cc,nn

Operation: IF cc TRUE: $(SP-1) \leftarrow PC_H$
$(SP-2) \leftarrow PC_L$, $PC \leftarrow nn$

Format:

Mnemonic: CALL **Operands:** cc, nn

Object Code:

1	1	cc	cc	cc	1	0	0

n	n	n	n	n	n	n	n

n	n	n	n	n	n	n	n

Note: The first of the two n operands in the assembled object code above is the least significant byte of the two-byte memory address.

Description:

If condition cc is true, this instruction pushes the current contents of the Program Counter (PC) onto the top of the external memory stack, then loads the operands nn into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into PC.) If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. The stack push is accomplished by first decrementing the current contents of the Stack Pointer (SP), loading the high-order byte of the PC contents into the memory address now pointed to by SP, then decrementing SP again, and loading the low-order byte of the PC contents into the top of the stack. **Note:** Because this is a three-byte instruction, the Program Counter will have been incremented by three before the push is executed. Condition cc is programmed as one of eight status bits which corresponds to condition bits in the Flag Register (register F). Those eight status bits are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code:

cc	Condition	Relevant Flag
000	NZ non-zero	Z (=0)
001	Z zero	Z (=1)
010	NC non-carry	C (=0)
011	C carry	C (=1)
100	PO parity odd	P/V (=0)
101	PE parity even	P/V (=1)
110	P sign positive	S (=0)
111	M sign negative	S (=1)

If cc is true:

M cycles: 5 T states: 17(4,3,4,3,3) 4 MHz E.T.: 4.25

If cc is false:

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the C Flag in the F register is reset, the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	D4H
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the three-byte instruction D43521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL NC, 2135H

After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

RET

RETurn

Operation: $PC_L \Leftarrow (SP)$, $PC_H \Leftarrow (SP+1)$

Format:

Mnemonic: RET **Operands:**

Object Code:

1	1	0	0	1	0	0	1

C9

Description:

Control is returned to the original program flow by popping the previous contents of the Program Counter (PC) off the top of the external memory stack, where they were pushed by the CALL instruction. This is accomplished by first loading the low-order byte of the PC with the contents of the memory address

pointed to by the Stack Pointer (SP), then incrementing the SP and loading the high-order byte of the PC with the contents of the memory address now pointed to by the SP. (The SP is now incremented a second time.) On the following machine cycle the CPU will fetch the next program opcode from the location in memory now pointed to by the PC.

M cycles: 3 T states: 10(4,3,3) 4 MHz E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

RET

the contents of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

Before:

Program Counter	Address	Stack
3535	2000	B5
	2001	18
	2002	2E
	2003	30

Stack Pointer
2000

After RET:

Program Counter	Address	Stack
18B5	2002	2E
	2003	30

Stack Pointer
2002

RET cc

RETurn

Operation: IF cc TRUE: $PC_L \Leftarrow (SP)$, $PC_H \Leftarrow (SP+1)$

Format:

Mnemonic: RET **Operands:** cc

Object Code:

1	1	cc	cc	cc	0	0	0

Description:

If condition cc is true, control is returned to the original program flow by popping the previous contents of the Program Counter (PC) off the top of the external memory stack, where they were pushed by the CALL instruction. This is accomplished by first loading the low-order byte of the PC with the contents of the memory address pointed to by the Stack Pointer (SP), then incrementing the SP, and loading the high-order byte of the PC with the contents of the memory address now pointed to by the SP. (The SP is now incremented a second time.) On the following machine cycle the CPU will fetch the next program opcode from the location in memory now pointed to by the PC. If condition cc is false, the PC is simply incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status bits which correspond to condition bits in the Flag Register F). These eight status bits are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code.

cc	Condition	Relevant Flag
000	NZ non-zero	Z (=0)
001	Z zero	Z (=1)
010	NC non-carry	C (=0)
011	C carry	C (=1)
100	PO parity odd	P/V (=0)
101	PE parity even	P/V (=1)
110	P sign positive	S (=0)
111	M sign negative	S (=1)

If cc is true:

M cycles: 3 T states: 11(5,3,3) 4 MHz E.T.: 2.75

If cc is false:

M cycles: 1 T states: 5 4 MHz E.T.: 1.25

Condition Bits Affected: None

Example:

If the S flag in the F register is set, the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

RET M

the contents of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

RETI

Operation: Return from interrupt

Format:

Mnemonic: RETI **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	0	1	4D

Description:

This instruction is used at the end of an interrupt service routine to:

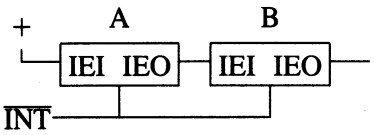
1. Restore the contents of the Program Counter (PC) (analogous to the RET instruction).
2. To signal an I/O device that the interrupt routine has been completed. The RETI instruction facilitates the nesting of interrupts, allowing higher priority devices to suspend service of lower priority service routines. This instruction also resets the IFF1 and IFF2 flip flops.

M cycles: 4 T states: 14(4,4,3,3) 4 MHz E.T.: 3.50

Condition Bits Affected: None

Example:

Given: Two interrupting devices, A and B, connected in a daisy chain configuration with A having a higher priority than B.



B generates an interrupt and is acknowledged. (The interrupt enable out, IEO, of B goes low, blocking any lower priority devices from interrupting while B is being serviced). Then A generates an interrupt, suspending service of B. (The

IEO of A goes 'low' indicating that a higher priority device is being serviced.) The A routine is completed and a RETI is issued resetting the IEO of A, allowing the B routine to continue. A second RETI is issued on completion of the B routine and the IEO of B is reset (high), allowing lower priority devices interrupt access.

RETN

Operation: Return from non maskable interrupt

Format:

Mnemonic: RETN **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	1	45

Description:

Used at the end of a service routine for a non maskable interrupt, this instruction executes an unconditional return which functions identically to the RET instruction. That is, the previously stored contents of the Program Counter (PC) are popped off the top of the external memory stack; the low-order byte of PC is loaded with the contents of the memory location pointed to by the Stack Pointer (SP), SP is incremented, the high-order byte of PC is loaded with the contents of the memory location now pointed to by SP, and SP is incremented again. Control is now returned to the original program flow: on the following machine cycle the CPU will fetch the next opcode from the location in memory now pointed to by the PC. Also the state of IFF2 is copied back into IFF1 to the state it had prior to the acceptance of the NMI.

M cycles: 4 T states: 14(4,4,3,3) 4 MHz E.T.: 3.50

Condition Bits Affected: None

Example:

If the contents of the Stack Pointer are 1000H and the contents of the Program Counter are 1A45H when a non maskable interrupt (NMI) signal is received, the CPU will ignore the next instruction and will instead restart to memory address 0066H. That is, the current Program Counter contents of 1A45H will be pushed onto the external stack address of 0FFFH and 0FFEH, high order byte first, and

0066H will be loaded onto the Program Counter. That address begins an interrupt service routine which ends with RETN instruction. Upon the execution of RETN, the former Program Counter contents are popped off the external memory stack, low-order first, resulting in a Stack Pointer contents again of 1000H. The program flow continues where it left off with an opcode fetch to address 1A45H.

RST p

ReSTart

Operation: $(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, $PC_H \leftarrow O$, $PC_L \leftarrow P$

Format:

Mnemonic: RST **Operands:** P

Object Code:

1	1	t	t	t	1	1	1

Description:

The current Program Counter (PC) contents are pushed onto the external memory stack, and the page zero memory location given by operand p is loaded into the PC. Program execution then begins with the opcode in the address now pointed to by PC. The push is performed by first decrementing the contents of the Stack Pointer (SP), loading the high-order byte of PC into the memory address now pointed to by SP, decrementing SP again, and loading the low-order byte of PC into the address now pointed to by SP. The ReSTart instruction allows for a Call to a subroutine at one of eight addresses as shown in the table below. The operand p is assembled into the object code using the t column of the table. **Note:** Since all addresses are in page zero of memory, the high order byte of PC is loaded with 00H. The number selected from the "p" column of the table is loaded into the low-order byte of PC.

At the end of the subroutine a RETurn instruction can be used to return to the original program by popping the top of the stack back into PC.

P	t
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

M cycles: 3 T states: 11(5,3,3) 4 MHz E.T.: 2.75

Example:

If the contents of the Program Counter are 15B3H, after the execution of

RST 18H (Object code 11011111)

the PC will contain 0018H, as the address of the next opcode to be fetched, and the top number on the stack will be 15B3H.

Input and Output Group

IN A,(n)

INput

Operation: A ⇦ (n)

Format:

Mnemonic: IN **Operands:** A, (n)

Object Code:

1	1	0	1	1	0	1	1	DB
n	n	n	n	n	n	n	n	

Description:

The number of the input port is n. Data is input to register A. The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into the Accumulator (register A) in the CPU.

M cycles: 3 T states: 11(4,3,4) 4 MHz E.T.: 2.75

Condition Bits Affected: None

Example:

If the contents of the Accumulator are 23H and the byte 7BH is available at the peripheral device mapped to I/O port address 01H, then after the execution of

IN A,(01H)

the Accumulator will contain 7BH.

IN r,(C)

INput

Operation: r ⇦ (C)

Format:

Mnemonic: IN **Operands:** r, (C)

Object Code:

1	1	1	0	1	1	0	1

ED

0	1	r	r	r	0	0	0

Description:

Register C contains the number of the input port. Data is input to register r. The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into register r in the CPU. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding three-bit "r" field for each. The flags will be affected, checking the input data.

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M cycles: 3 T states: 12(4,4,4) 4 MHz E.T.: 3.00

Condition Bits Affected:

S: Set if input data is negative; reset otherwise
Z: Set if input data is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

IN D,(C)

register D will contain 7BH

A typical use of the IN r, (C) instruction is for polled I/O. The following program continually polls or inputs data from port FF until a non-zero number appears. The program then reads in data from port FE. In this application, port FF is used as a data ready signal for port FE.

```
        LD      C, 0FFH      ; C points at port FF
LOOP    IN      B, (C)       ; input port FF to register B
        JR      Z, LOOP      ; continue polling until not zero
        IN      A, (0FEH)    ; input port FE to register A
```

INI

INput & Increment

Operation: (HL) ◊ (C), B ◊ B − 1, HL ◊ HL + 1

Format:

Mnemonic: INI **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	0	A2

Description:

Register C contains the number of the input port. Data input is placed in memory at the address pointed at by HL. The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are then placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter is decremented and register pair HL is incremented.

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S:	Unknown
Z:	Set if B − 1 = 0; reset otherwise
H:	Unknown
P/V:	Unknown
N:	Set
C:	Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

INI

memory location 1000H will contain 7BH, the HL register pair will contain 1001H, and register B will contain 0FH.

The following program will input data from input ports 1 through 80 and place the data into a buffer in memory.

```
        LD      B, 80
        LD      C,0
        LD      HL, BUFF
LOOP    INC     C
        INI
        JP      NZ, LOOP
```

INIR

INput Increment & Repeat

Operation: (HL) ⇦ (C), B ⇦ B − 1, HL ⇦ HL + 1

Format:

Mnemonic: INIR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED

1	0	1	1	0	0	1	0	B2

Description:

Register C contains the number of the input port. The data input is placed in memory at the address pointed at by the HL register pair. The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then register pair HL is incremented, the byte counter is decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input. Also interrupts will be recognized after each data transfer.

If B ≠ 0:

M cycles: 5 T states: 21(4,5,3,4,5) 4 MHz E.T.: 5.25

If B = 0:

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S:	Unknown
Z:	Set
H:	Unknown
P/V:	Unknown
N:	Set
C:	Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port of address 07H:

51H
A9H
03H

then after the execution of

INIR

the HL register pair will contain 1003H, register B will contain zero, and memory locations will have contents as follows:

Location	Contents
1000H	51H
1001H	A9H
1002H	03H

Here is a program to input 80 bytes from I/O port number FF and put them into an 80-byte buffer starting at address BUFF.

```
LD      HL, BUFF     ; HL points at first byte of buffer
LD      B, 80        ; load byte counter
LD      C, OFFH      ; port FF
IN IR                ; input 80 bytes
```

Note: this assumes that the input port can be synchronized with the input instructions.

IND

INput & Decrement

Operation: (HL) ◊ (C), B ◊ B − 1, HL ◊ HL − 1

Format:

Mnemonic: IND **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	1	0	AA

Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter and register pair HL are decremented.

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Unknown
Z: Set if B − 1 = 0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

IND

memory location 1000H will contain 7BH, the HL register pair will contain 0FFFH, and register B will contain 0FH.

INDR

INput Decrement & Repeat

Operation: (HL) ◊ (C), B ◊ B − 1, HL ◊ HL − 1

Format:

Mnemonic: INDR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	1	0	BA

Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then HL and the byte counter are decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input. Also interrupts will be recognized after each data transfer.

If B ≠ 0:

M cycles: 5 T states: 21(4,5,3,4,5) 4 MHz E.T.: 5.25

If B = 0:

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S:	Unknown
Z:	Set
H:	Unknown
P/V:	Unknown
N:	Set
C:	Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port address 07H:

51H
A9H
03H

then after the execution of

INDR

the HL register pair will contain 0FFDH, register B will contain zero, and memory locations will have contents as follows:

Location	Contents
0FFEH	03H
0FFFH	A9H
1000H	51H

OUT (n),A

OUTput

Operation: (n) ⇦ A

Format:

Mnemonic: OUT **Operands:** (n), A

Object Code:

1	1	0	1	0	0	1	1

D3

n	n	n	n	n	n	n	n

Description:

The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator (register A) also appear on the top half (A8 through A15) of the address bus at this time. Then the byte contained in the Accumulator is placed on the data bus and written into the selected peripheral device.

M cycles: 3 T states: 11(4,3,4) 4 MHz E.T.: 2.75

Condition Bits Affected: None

Example:

If the contents of the Accumulator are 23H, then after the execution of

OUT 01H,A

the byte 23H will have been written to the peripheral device mapped to I/O port address 01H.

OUT (C),r

OUTput

Operation: (C) ◊ r

Format:

Mnemonic: OUT **Operands:** (C), r

Object Code:

1	1	1	0	1	1	0	1

ED

0	1	r	r	r	0	0	1

Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then the byte contained in register r is placed on the data bus and written into the selected peripheral device. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding three-bit "r" field for each which appears in the assembled object code:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M cycles: 3 T states: 12(4,4,4) 4 MHz E.T.: 3.00

Condition Bits Affected: None

Example:

If the contents of register C are 01H and the contents of register D are 5AH, after the execution of

OUT (C),D

the byte 5AH will have been written to the peripheral device mapped to I/O port address 01H.

OUTI

OUTput & Increment

Operation: (C) ⇦ (HL), B ⇦ B − 1, HL ⇦ HL + 1

Format:

Mnemonic: OUTI **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	1	A3

Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through

A15) of the address bus. The byte to be output is placed on the data bus and written into selected peripheral device. Finally the register pair HL is incremented.

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Unknown
Z: Set if B − 1 = 0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory address 1000H are 59H, then after the execution of

OUTI

register B will contain 0FH, the HL register pair will contain 1001H, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

OTIR

OuTput Increment & Repeat

Operation: (C) ◊ (HL), B ◊ B − 1, HL ◊ HL + 1

Format:

Mnemonic: OTIR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	1	1	B3

Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus

to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is incremented. If the decremented B register is not zero, the Program Counter (PC) is decremented by two and the instruction is repeated. If B has gone to zero, the instruction is terminated. Note that if B is set to zero prior to instruction execution, the instruction will output 256 bytes of data. Also, interrupts will be recognized after each data transfer.

If B ≠ Ø:

M cycles: 5 T states: 21(4,5,3,4,5) 4 MHz E.T.: 5.25

If B = Ø:

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.ØØ

Condition Bits Affected:

S:	Unknown
Z:	Set
H:	Unknown
P/V:	Unknown
N:	Set
C:	Not affected

Example:

If the contents of register C are Ø7H, the contents of register B are Ø3H, the contents of the HL register pair are 1ØØØH, and memory locations have the following contents:

Location	Contents
1ØØØH	51H
1ØØ1H	A9H
1ØØ2H	Ø3H

then after the execution of

OTIR

the HL register pair will contain 1ØØ3H, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address Ø7H in the following sequence:

51H
A9H
Ø3H

OUTD

OUTput & Decrement

Operation: (C) ◊ (HL), B ◊ B − 1, HL ◊ HL − 1

Format:

Mnemonic: OUTD **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	1	1	AB

Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Finally the register pair HL is incremented.

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Unknown
Z: Set if B − 1 = 0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory location 1000H are 59H, after the execution of

OUTD

register B will contain 0FH, the HL register pair will contain 0FFFH, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

OTDR

OUTput Decrement & Repeat

Operation: (C) ◊ (HL), B ◊ B − 1, HL ◊ HL − 1

Format:

Mnemonic: OTDR **Operands:**

Object Code:

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	1	1	BB

Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is decremented and if the decremented B register is not zero, the Program Counter (PC) is decremented by 2 and the instruction is repeated. If B has gone to zero, the instruction is terminated. Note that if B is set to zero prior to instruction execution, the instruction will output 256 byte of data. Also, interrupts will be recognized after each data transfer.

If B ≠ 0:

M cycles: 5 T states: 21(4,5,3,4,5) 4 MHz E.T.: 5.25

If B = 0:

M cycles: 4 T states: 16(4,5,3,4) 4 MHz E.T.: 4.00

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location	Contents
0FFEH	51H
0FFFH	A9H
1000H	03H

then after the execution of

OTDR

the HL register pair will contain 0FFDH, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address 07H in the following sequence:

03H
A9H
51H

Appendix A / Using the TPSRC Utility (Disk Systems Only)

This utility allows disk systems to:

A. Read the source tapes created by the tape version of the Editor/Assembler, and copy these to disk.

B. Copy a disk object file (machine-language program) onto tape in the "SYSTEM" format.

Under TRSDOS READY, type TPSRC (ENTER). The program will start and ask you to select either (1) source tape input or (2) object tape output.

Source Tape Input

If you type 1 (ENTER), the program will tell you to get the recorder ready. Get your recorder ready to play the source tape (created by the W command of the Tape Editor/Assembler). Then press (ENTER).

TPSRC will read the tape and create a disk file with the same name as the tape and with the extension /SRC. The resultant file may be loaded by the Disk Editor/Assembler (L command).

Object Tape Output

If you type 2 (ENTER), the program will ask you for the name of the disk file. (The file must be in the correct program format, as created by the Disk Editor/Assembler A command.) Type in the file name and press (ENTER).

Next, TPSRC will prompt you to get the recorder ready. Using a blank tape, prepare the recorder to record. Then press (ENTER). TPSRC will then write out the object tape. The object tape will be given the name of the disk object file.

The resultant tape is in the SYSTEM format, and may be loaded according to the instructions in Section 5.

Appendix B / Model I Subroutines

These are subroutines which are in the Read Only Memory (ROM) of your Model I Level I or Level II BASIC Computer. You can call them using an assembly language program.

The left-hand column lists the subroutines. The next columns demonstrate example assembly language programs which call these subroutines.

If you have a Model I disk system, you can also call subroutines which are a part of your TRS-80 Disk Operating System (TRSDOS). These are listed in your Model I "TRSDOS Disk BASIC Reference Manual."

The Model III BASIC subroutines are listed in the "TRS-80 Model III Operation and BASIC Language Reference Manual." (See the Appendix of the Operation Section.) The Model III TRSDOS subroutines are in the "Technical Information" of the "Model III Disk System's Owners Manual."

Level I BASIC Subroutines

KEYBOARD SCAN A-register contains input byte; input byte is displayed at current cursor.	WAIT	CALL JR	0B40H Z,WAIT	;SCAN ;Z= 1 IF KB CLEAR
DISPLAY BYTE AT CURSOR		PUSH PUSH LD RST POP POP	DE IY A,20H 10H IY DE	;MUST SAVE ; DE & IY ;BYTE TO DISPLAY ;DISPLAY BYTE ;RESTORE ; DE & IY
TURN ON CASSETTE On board cassette is turned on via remote plug		CALL	0FE9H	;TURN ON CASSETTE
SAVE MEMORY TO CASSETTE Cassette is turned off		CALL LD LD CALL	0FE9H HL,700H DE,7100H 0F4BH	;TURN ON CASSETTE ;START ADDRESS ;LAST+1 ADDRESS ;SAVE IT

LOAD MEMORY FROM CASSETTE On return HL = last + 1 address Z = 0 if checksum error Z = 1 if checksum OK Cassette is turned off		CALL	0EF4H	;TURN ON & READ
RETURN TO LEVEL I BASIC		Press Reset		
		JP	0	;POWER UP
		JP	01C9H	;RE-ENTRY WITH READY

Level II BASIC Subroutines

TURN ON CURSOR CHARACTER		PUSH	DE	;MUST SAVE
		PUSH	IY	; DE & IY
		LD	A,0EH	;0EH IS CURSOR BYTE
		CALL	33H	;DISPLAY ROUTINE
		POP	IY	;RESTORE
		POP	DE	; DE & IY
KEYBOARD SCAN A-register contains byte when loop falls through. Byte is not displayed on Screen!		PUSH	DE	;MUST SAVE
		PUSH	IY	; DE & IY
	AGN	CALL	2BH	;SCAN ROUTINE
		OR	A	;A=0 IF KB CLEAR
		JR	Z,AGN	;BRANCH IF NO BYTE
		POP	IY	;RESTORE
		POP	DE	; DE & IY
DISPLAY BYTE AT CURSOR		PUSH	DE	;MUST SAVE
		PUSH	IY	; DE & IY
		LD	A,20H	;BYTE TO DISPLAY
		CALL	33H	;DISPLAY
		POP	IY	;RESTORE
		POP	DE	; DE & IY
	;A-REGISTER SPECIFIES CASSETTE (0 OR 1)			
DEFINE DRIVE		LD	A,0	;ON BOARD CASSETTE
		CALL	0212H	;DEFINE DRIVE
WRITE LEADER AND SYNC BYTE		CALL	0287H	
TURN OFF CASSETTE		CALL	01F8H	

SAVE MEMORY TO CASSETTE

User must CALL 264H often enough to keep up with 500 baud. Timing is automatic.

```
        LD      A,0         ;ON BOARD CASSETTE
        CALL    0212H       ;DEFINE DRIVE
        CALL    0287H       ;WRITE LEADER
        LD      A,20H       ;BYTE TO RECORD
        CALL    0264H       ;OUTPUT BYTE

        CALL    01F8H       ;CASSETTE OFF
```

LOOK FOR LEADER AND SYNC BYTE

```
        CALL    0296H
```

LOAD MEMORY FROM CASSETTE

Your program must CALL 0235H often enough to keep up with 500 baud, and must do its own checksum if desired. A-register contains byte read. The user must turn off the cassette (CALL 01F8H) when all bytes have been read.

```
        LD      A,0         ;DEFINE DRIVE
        CALL    0212H       ;FIND SYNC BYTE
        CALL    0296H       ;READ ONE BYTE
        CALL    0235H
```

RETURN TO LEVEL II BASIC

```
        Press RESET
        JP      0           ;LIKE POWER UP
        JP      1A19H       ;RE-ENTRY
```

OUTPUT TO LINE PRINTER (LEVEL II ONLY)

```
                            ;PUT ASCII BYTE IN
                            ;A-REGISTER AND CALL
                             PRTOUT
                            ;BUSY CONDITION TESTED
                             FOR
PRTOUT  EXX                 ;SAVE REGS.
        LD      HL,37E8H    ;LOAD LP POINTER
                             IN HL
PRTLP8  LD      D,(HL)      ;LOAD LP STATUS BYTE
        BIT     7,D         :IS THE PRINTER
        JP      NZ,PRTLP8    BUSY?
        LD      (HL),A
        EXX                 ;OUTPUT BYTE TO
        RET                  PRINTER
```

Appendix C / Z-80 Status Indicators (Flags)

The flag register (F and F') supplies information to the user regarding the status of the Z-80 at any given time. The bit positions for each flag are shown below:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

WHERE:

C = CARRY FLAG
N = ADD/SUBTRACT FLAG
P/V = PARITY/OVERFLOW FLAG
H = HALF-CARRY FLAG
Z = ZERO FLAG
S = SIGN FLAG
X = NOT USED

Each of the two Z-80 Flag Registers contains 6 bits of status information which are set or reset by CPU operations. (Bits 3 and 5 are not used.) Four of these bits are testable (C,P/V,Z and S) for use with conditional jump, call or return instructions. Two flags are not testable (H,N) and are used for BCD arithmetic.

Carry Flag (C)

The carry bit is set or reset depending on the operation begin performed. For 'ADD' instructions that generate a carry and 'SUBTRACT' instructions that generate no borrow, the Carry Flag will be set. The Carry Flag is reset by an ADD that does not generate a carry and a 'SUBTRACT' that generates a borrow. This saved carry facilitates software routines for extended precision arithmetic. Also, the 'DAA' instruction will set the Carry Flag if the conditions for making the decimal adjustment are met.

For instructions RLA, RRA, RLS and RRS, the carry bit is used as a link between the LSB and MSB for any register or memory location. During instructions RLCA, RLC'S and SLA'S, the carry contains the last value shifted out of bit 7 of any register or memory location. During instructions RRCA, RRC'S, SRA'S and SRL'S the carry contains the last value shifted out of bit 0 of any register or memory location.

For the logical instructions AND'S, OR'S and XOR'S, the carry will be reset.

The Carry Flag can also be set (SCF) and complemented (CCF).

Add/Subtract Flag (N)

This flag is used by the decimal adjust accumulator instruction (DAA) to distingiush between 'ADD' and 'SUBTRACT, instructions. For all 'ADD' instructions, N will be set to a '0.' For all 'SUBTRACT' instructions, N will be set to a "1."

Parity/Overflow Flag (P/V)

This flag is set to a particular state depending on the operation being performed.

For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (−128). This overflow condition can be determined by examining the sign bits of the operands.

For addition, operands with different signs will never cause overflow. When adding operands with like signs and the result has a different sign, the overflow flag is set. For example:

+120 =	0111 1000	ADDEND
+105 =	0110 1001	AUGEND
+225	1110 0001	(−95) SUM

The two numbers added together has resulted in a number that exceeds +127 and the two positive operands has resulted in a negative number (−95) which is incorrect. The overflow flag is therefore set.

For subtraction, overflow can occur for operands of unlike signs. Operands of like sign will never cause overflow. For example:

+127	0111 1111	MINUEND
(−) −64	1100 0000	SUBTRAHEND
+191	1011 1111	DIFFERENCE

The minuend sign has changed from a positive to a negative, giving an incorrect difference. Overflow is therefore set.

Another method for predicting an overflow is to observe the carry into and out of the sign bit. If there is a carry in and no carry out, or if there is no carry in and a carry out, then overflow has occurred.

This flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of '1' bits in a byte are counted. If the total is odd, 'ODD' parity (P=0) is flagged. If the total is even, 'EVEN' parity is flagged (P=1).

During search instructions (CPI,CPIR,CPD,CPDR) and block transfer instructions (LDI,LDIR,LDD,LDDR) the P/V flag monitors the state of the byte count register (BC). When decrementing, the byte counter results in a zero value, the flag is reset to 0, otherwise the flag is a Logic 1.

During LD A,I and LD A,R instructions, the P/V flag will be set with the contents of the interrupt enable flip-flop (IFF2) for storage or testing.

When inputting a byte from an I/O device, IN r,(C), the flag will be adjusted to indicate the parity of the data.

The Half Carry Flag (H)

The Half Carry Flag (H) will be set or reset depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation. This flag is used by

the decimal adjust accumulator instruction (DAA) to correct the result of a packed BCD add or subtract operation. The H flag will be set (1) or rest (0) according to the following table:

H	ADD	SUBTRACT
1	There is a carry from Bit 3 to Bit 4	There is no borrow from Bit 4
0	There is no carry from Bit 3 to Bit 4	There is a borrow from Bit 4

The Zero Flag (Z)

The Zero Flag (Z) is set or reset if the result generated by the execution of certain instructions is a zero.

For 8-bit arithmetic and logical operations, the Z flag will be set to a '1' if the resulting byte in the Accumulator is zero. If the byte is not zero, the Z flag is reset to '0.'

For compare (search) instructions, the Z flag will be set to a '1' if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL.

When testing a bit in a register or memory location, the Z flag will contain the complemented state of the indicated bit (see Bit b,s).

When inputting or outputting a byte between a memory location and an I/O device (INI;IND;OUTI and OUTD), if the result of B-1 is zero, the Z flag is set, otherwise it is reset. Also for byte inputs from I/O devices using IN r,(C), the Z Flag is set to indicate a zero byte input.

The Sign Flag (S)

The Sign Flag (S) stores the state of the most significant bit of the Accumulator (Bit 7). When the Z80 performs arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. A positive number is identified by a '0' in bit 7. A negative number is identified by a '1'. The binary equivalent of the magnitude of a positive number is stored in bits 0 to 6 for a total range of from 0 to 127. A negative number is represented by the two's complement of the equivalent positive number. The total range for negative numbers is from -1 to -128.

When inputting a byte from a I/O device to a register, IN r,(C) the S flag will indicate either positive (S = 0) or negative (S = 1) data.

Appendix D
Numeric List of Instruction Set

Following is a listing of object codes in numerical order in column two followed by the nmemonic or source statement in column four.

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0000	00	1	NOP
0001	018405	2	LD BC,NN
0004	02	3	LD (BC),A
0005	03	4	INC BC
0006	04	5	INC B
0007	05	6	DEC B
0008	0620	7	LD B,N
000A	07	8	RLCA
000B	08	9	EX AF,AF'
000C	09	10	ADD HL,BC
000D	0A	11	LD A,(BC)
000E	0B	12	DEC BC
000F	0C	13	INC C
0010	OD	14	DEC C
0011	0E20	15	LD C,N
0013	0F	16	RRCA
0014	102E	17	DJNZ DIS
0016	118405	18	LD DE,NN
0019	12	19	LD (DE),A
001A	13	20	INC DE
001B	14	21	INC D
001C	15	22	DEC D
001D	1620	23	LD D,N
001F	17	24	RLA
0020	182E	25	JR DIS
0022	19	26	ADD HL,DE
0023	1A	27	LD A,(DE)
0024	1B	28	DEC DE
0025	1C	29	INC E
0026	ID	30	DEC E
0027	1E20	31	LD E,N
0029	1F	32	RRA
002A	202E	33	JR NZ,DIS
002C	218405	34	LD HL,NN
002F	228405	35	LD (NN),HL
0032	23	36	INC HL
0033	24	37	INC H
0034	25	38	DEC H
0035	2620	39	LD H,N
0037	27	40	DAA
0038	282E	41	JR Z,DIS
003A	29	42	ADD HL,HL
003B	2A8405	43	LD HL,(NN)
003E	2B	44	DEC HL
003F	2C	45	INC L
0040	2D	46	DEC L
0041	2E20	47	LD L,N
0043	2F	48	CPL
0044	302E	49	JR NC,DIS
0046	318405	50	LD SP,NN
0049	328405	51	LD (NN),A
004C	33	52	INC SP
004D	34	53	INC (HL)
004E	35	54	DEC (HL)
004F	3620	55	LD (HL),N
0051	37	56	SCF
0052	382E	57	JR C,DIS
0054	39	58	ADD HL,SP
0055	3A8405	59	LD A,(NN)
0058	3B	60	DEC SP
0059	3C	61	INC A
005A	3D	62	DEC A
005B	3E20	63	LD A,N
005D	3F	64	CCF
005E	40	65	LD B,B
005F	41	66	LD B,C
0060	42	67	LD B,D
0061	43	68	LD B,E
0062	44	69	LD B,H(NN)
0063	45	70	LD B,L
0064	46	71	LD B,(HL)
0065	47	72	LD B,A
0066	48	73	LD C,B
0067	49	74	LD C,C
0068	4A	75	LD C,D
0069	4B	76	LD C,E
006A	4C	77	LD C,H
006B	4D	78	LD C,L
006C	4E	79	LD C,(HL)
006D	4F	80	LD C,A
006E	50	81	LD D,B
006F	51	82	LD D,C
0070	52	83	LD D,D
0071	53	84	LD D,E
0072	54	85	LD D,H
0073	55	86	LD D,L
0074	56	87	LD D,(HL)
0075	57	88	LD D,A
0076	58	89	LD E,B
0077	59	90	LD E,C
0078	5A	91	LD E,D
0079	5B	92	LD E,E
007A	5C	93	LD E,H
007B	5D	94	LD E,L
007C	5E	95	LD E,(HL)
007D	5F	96	LD E,A
007E	60	97	LD H,B
007F	61	98	LD H,C
0080	62	99	LD H,D
0081	63	100	LD H,E
0082	64	101	LD H,H
0083	65	102	LD H,L
0084	66	103	LD H,(HL)
0085	67	104	LD H,A
0086	68	105	LD L,B
0087	69	106	LD L,C

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0088	6A	107	LD L,D
0089	6B	108	LD L,E
008A	6C	109	LD L,H
008B	6D	110	LD L,L
008C	6E	111	LD L,(HL)
008D	6F	112	LD L,A
008E	70	113	LD (HL),B
008F	71	114	LD (HL),C
0090	72	115	LD (HL),D
0091	73	116	LD (HL),E
0092	74	117	LD (HL),H
0093	75	118	LD (HL),L
0094	76	119	HALT
0095	77	120	LD (HL),A
0096	78	121	LD A,B
0097	79	122	LD A,C
0098	7A	123	LD A,D
0099	7B	124	LD A,E
009A	7C	125	LD A,H
009B	7D	126	LD A,L
009C	7E	127	LD A,(HL)
009D	7F	128	LD A,A
009E	80	129	ADD A,B
009F	81	130	ADD A,C
00A0	82	131	ADD A,D
00A1	83	132	ADD A,E
00A2	84	133	ADD A,H
00A3	85	134	ADD A,L
00A4	86	135	ADD A,(HL)
00A5	87	136	ADD A,A
00A6	88	137	ADC A,B
00A7	89	138	ADC A,C
00A8	8A	139	ADC A,D
00A9	8B	140	ADC A,E
00AA	8C	141	ADC A,H
00AB	8D	142	ADC A,L
00AC	8E	143	ADC A,(HL)
00AD	8F	144	ADC A,A
00AE	90	145	SUB B
00AF	91	146	SUB C
00B0	92	147	SUB D
00B1	93	148	SUB E
00B2	94	149	SUB H
00B3	95	150	SUB L
00B4	96	151	SUB (HL)
00B5	97	152	SUB A
00B6	98	153	SBC A,B
00B7	99	154	SBC A,C
00B8	9A	155	SBC A,D
00B9	9B	156	SBC A,E
00BA	9C	157	SBC A,H
00BB	9D	158	SBC A,L
00BC	9E	159	SBC A,(HL)
00BD	9F	160	SBC A,A
00BE	A0	161	AND B
00BF	A1	162	AND C
00C0	A2	163	AND D
00C1	A3	164	AND E
00C2	A4	165	AND H
00C3	A5	166	AND L
00C4	A6	167	AND (HL)
00C5	A7	168	AND A
00C6	A8	169	XOR B
00C7	A9	170	XOR C
00C8	AA	171	XOR D
00C9	AB	172	XOR E
00CA	AC	173	XOR H
00CB	AD	174	XOR L
00CC	AE	175	XOR (HL)
00CD	AF	176	XOR A
00CE	B0	177	OR B
00CF	B1	178	OR C
00D0	B2	179	OR D
00D1	B3	180	OR E
0002	B4	181	OR H
00D3	B5	182	OR L
00D4	B6	183	OR (HL)
00D5	B7	184	OR A
00D6	B8	185	CP B
00D7	B9	186	CP C
00D8	BA	187	CP D
00D9	BB	188	CP E
00DA	BC	189	CP H
00DB	BD	190	CP L
00DC	BE	191	CP (HL)
00DD	BF	192	CP A
00DE	C0	193	RET NZ
00DF	C1	194	POP BC
00E0	C28405	195	JP NZ, NN
00E3	C38405	196	JP NN
00E6	C48405	197	CALL NZ,NN
00E9	C5	198	PUSH BC
00EA	C620	199	ADD A,N
00EC	C7	200	RST 0
00ED	C8	201	RET Z
00EE	C9	202	RET
00EF	CA8405	203	JP Z,NN
00F2	CC8405	204	CALL Z,NN
00F5	CD8405	205	CALL NN
00F8	CE20	206	ADC A,N
00FA	CF	207	RST 8
00FB	D0	208	RET NC
00FC	D1	209	POP DE
00FD	D28405	210	JP NC,NN
0100	D320	211	OUT ,NA
0102	D48405	212	CALL NC,NN
0105	D5	213	PUSH DE
0106	D620	214	SUB N
0108	D7	215	RST 10H
0109	D8	216	RET C
010A	D9	217	EXX
010B	DA8405	218	JP C,NN
010E	DB20	219	IN A,N
0110	DC8405	220	CALL C,NN
0113	DE20	221	SBC A,N
0115	DF	222	RST 18H
0116	E0	223	RET PO
0117	E1	224	POP HL
0118	E28405	225	JP PO,NN
011B	E3	226	EX (SP),HL
011C	E48405	227	CALL PO,NN
011F	E5	228	PUSH HL

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0120	E620	229	AND N
0122	E7	230	RST 20H
0123	E8	231	RET PE
0124	E9	232	JP (HL)
0125	EA8405	233	JP PE,NN
0128	EB	234	EX DE,HL
0129	EC8405	235	CALL PE,NN
012C	EE20	236	XOR N
012E	EF	237	RST 28H
012F	F0	238	RET P
0130	F1	239	POP AF
0131	F28405	240	JP P,NN
0134	F3	241	DI
0135	F48405	242	CALL P,NN
0138	F5	243	PUSH AF
0139	F620	244	OR N
013B	F7	245	RST 30H
013C	F8	246	RET M
013D	F9	247	LD SP,HL
013E	FA8405	248	JP M,NN
0141	FB	249	EI
0142	FC8405	250	CALL M,NN
0145	FE20	251	CP N
0147	FF	252	RST 38H
0148	CB00	253	RLC B
014A	CB01	254	RLC C
014C	CB02	255	RLC D
014E	CB03	256	RLC E
0150	CB04	257	RLC H
0152	CB05	258	RLC L
0154	CB06	259	RLC (HL)
0156	CB07	260	RLC A
0158	CB08	261	RRC B
015A	CB09	262	RRC C
015C	CB0A	263	RRC D
015E	CB0B	264	RRC E
0160	CB0C	265	RRC H
0162	CB0D	266	RRC L
0164	CB0E	267	RRC (HL)
0166	CB0F	268	RRC A
0168	CB10	269	RL B
016A	CB11	270	RL C
016C	CB12	271	RL D
016E	CB13	272	RL E
0170	CB14	273	RL H
0172	CB15	274	RL L
0174	CB16	275	RL (HL)
0176	CB17	276	RL A
0178	CB18	277	RR B
017A	CB19	278	RR C
017C	CB1A	279	RR D
017E	CB1B	280	RR E
0180	CB1C	281	RR H
0182	CB1D	282	RR L
0184	CB1E	283	RR (HL)
0186	CB1F	284	RR A
0188	CB20	285	SLA B
018A	CB21	286	SLA C
018C	CB22	287	SLA D
018E	CB23	288	SLA E
0190	CB24	289	SLA H

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0192	CB25	290	SLA L
0194	CB26	291	SLA (HL)
0196	CB27	292	SLA A
0198	CB28	293	SRA B
019A	CB29	294	SRA C
019C	CB2A	295	SRA D
019E	CB2B	296	SRA E
01A0	CB2C	297	SRA H
01A2	CB2D	298	SRA L
01A4	CB2E	299	SRA (HL)
01A6	CB2F	300	SRA A
01A8	CB38	301	SRL B
01AA	CB39	302	SRL C
01AC	CB3A	303	SRL D
01AE	CB3B	304	SRL E
01B0	CB3C	305	SRL H
01B2	CB3D	306	SRL L
01B4	CB3E	307	SRL (HL)
01B6	CB3F	308	SRL A
01B8	CB40	309	BIT 0,B
01BA	CB41	310	BIT 0,C
01BC	CB42	311	BIT 0,D
01BE	CB43	312	BIT 0,E
01C0	CB44	313	BIT 0,H
01C2	CB45	314	BIT 0,L
01C4	CB46	315	BIT 0,(HL)
01C6	CB47	316	BIT 0,A
01C8	CB48	317	BIT 1,B
01CA	CB49	318	BIT 1,C
01CC	CB4A	319	BIT 1,D
01CE	CB4B	320	BIT 1,E
01D0	CB4C	321	BIT 1,H
01D2	CB4D	322	BIT 1,L
01D4	CB4E	323	BIT 1,(HL)
01D6	CB4F	324	BIT 1,A
01D8	CB50	325	BIT 2,B
01DA	CB51	326	BIT 2,C
01DC	CB52	327	BIT 2,D
01DE	CB53	328	BIT 2,E
01E0	CB54	329	BIT 2,H
01E2	CB55	330	BIT 2,L
01E4	CB56	331	BIT 2,(HL)
01E6	CB57	332	BIT 2,A
01E8	CB58	333	BIT 3,B
01EA	CB59	334	BIT 3,C
01EC	CB5A	335	BIT 3,D
01EE	CB5B	336	BIT 3,E
01F0	CB5C	337	BIT 3,H
01F2	CB5D	338	BIT 3,L
01F4	CB5E	339	BIT 3,(HL)
01F6	CB5F	340	BIT 3,A
01F8	CB60	341	BIT 4,B
01FA	CB61	342	BIT 4,C
01FC	CB62	343	BIT 4,D
01FE	CB63	344	BIT 4,E
0200	CB64	345	BIT 4,H
0202	CB65	346	BIT 4,L
0204	CB66	347	BIT 4,(HL)
0206	CB67	348	BIT 4,A
0208	CB68	349	BIT 5,B
020A	CB69	350	BIT 5,C

LOC	OBJ CODE	STMT	SOURCE STATEMENT
020C	CB6A	351	BIT 5,D
020E	CB6B	352	BIT 5,E
0210	CB6C	353	BIT 5,H
0212	CB6D	354	BIT 5,L
0214	CB6E	355	BIT 5,(HL)
0216	CB6F	356	BIT 5,A
0218	CB70	357	BIT 6,B
021A	CB71	358	BIT 6,C
021C	CB72	359	BIT 6,D
021E	CB73	360	BIT 6,E
0220	CB74	361	BIT 6,H
0222	CB75	362	BIT 6,L
0224	CB76	363	BIT 6,(HL)
0226	CB77	364	BIT 6,A
0228	CB78	365	BIT 7,B
022A	CB79	366	BIT 7,C
022C	CB7A	367	BIT 7,D
022E	CB7B	368	BIT 7,E
0230	CB7C	369	BIT 7,H
0232	CB7D	370	BIT 7,L
0234	CB7E	371	BIT 7,(HL)
0236	CB7F	372	BIT 7,A
0238	CB80	373	RES 0,B
023A	CB81	374	RES 0,C
023C	CB82	375	RES 0,D
023E	CB83	376	RES 0,E
0240	CB84	377	RES 0,H
0242	CB85	378	RES 0,L
0244	CB86	379	RES 0,(HL)
0246	CB87	380	RES 0,A
0248	CB88	381	RES 1,B
024A	CB89	382	RES 1,C
024C	CB8A	383	RES 1,D
024E	CB8B	384	RES 1,E
0250	CB8C	385	RES 1,H
0252	CB8D	386	RES 1,L
0254	CB8E	387	RES 1,(HL)
0256	CB8F	388	RES 1,A
0258	CB90	389	RES 2,B
025A	CB91	390	RES 2,C
025C	CB92	391	RES 2,D
025E	CB93	392	RES 2,E
0260	CB94	393	RES 2,H
0262	CB95	394	RES 2,L
0264	CB96	395	RES 2,(HL)
0266	CB97	396	RES 2,A
0268	CB98	397	RES 3,B
026A	CB99	398	RES 3,C
026C	CB9A	399	RES 3,D
026E	CB9B	400	RES 3,E
0270	CB9C	401	RES 3,H
0272	CB9D	402	RES 3,L
0274	CB9E	403	RES 3,(HL)
0276	CB9F	404	RES 3,A
0278	CBA0	405	RES 4,B
027A	CBA1	406	RES 4,C
027C	CBA2	407	RES 4,D
027E	CBA3	408	RES 4,E
0280	CBA4	409	RES 4,H
0282	CBA5	410	RES 4,L
0284	CBA6	411	RES 4,(HL)

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0286	CBA7	412	RES 4,A
0288	CBA8	413	RES 5,B
028A	CBA9	414	RES 5,C
028C	CBAA	415	RES 5,D
028E	CBAB	416	RES 5,E
0290	CBAC	417	RES 5,H
0292	CBAD	418	RES 5,L
0294	CBAE	419	RES 5,(HL)
0296	CBAF	420	RES 5,A
0298	CBB0	421	RES 6,B
029A	CBB1	422	RES 6,C
029C	CBB2	423	RES 6,D
029E	CBB3	424	RES 6,E
02A0	CBB4	425	RES 6,H
02A2	CBB5	426	RES 6,L
02A4	CBB6	427	RES 6,(HL)
02A6	CBB7	428	RES 6,A
02A8	CBB8	429	RES 7,B
02AA	CBB9	430	RES 7,C
02AC	CBBA	431	RES 7,D
02AE	CBBB	432	RES 7,E
0280	CBBC	433	RES 7,H
0282	CBBD	434	RES 7,L
0284	CBBE	435	RES 7,(HL)
0286	CBBF	436	RES 7,A
0288	CBC0	437	SET 0,B
02BA	CBC1	438	SET 0,C
02BC	CBC2	439	SET 0,D
02BE	CBC3	440	SET 0,E
02C0	CBC4	441	SET 0,H
02C2	CBC5	442	SET 0,L
02C4	CBC6	443	SET 0,(HL)
02C6	CBC7	444	SET 0,A
02C8	CBC8	445	SET 1,B
02CA	CBC9	446	SET 1,C
02CC	CBCA	447	SET 1,D
02CE	CBCB	448	SET 1,E
02D0	CBCC	449	SET 1,H
02D2	CBCD	450	SET 1,L
02D4	CBCE	451	SET 1,(HL)
02D6	CBCF	452	SET 1,A
02D8	CBD0	453	SET 2,B
02DA	CBD1	454	SET 2,C
02DC	CBD2	455	SET 2,D
02DE	CBD3	456	SET 2,E
02E0	CBD4	457	SET 2,H
02E2	CBD5	458	SET 2,L
02E4	CBD6	459	SET 2,(HL)
02E6	CBD7	460	SET 2,A
02E8	CBD8	461	SET 3,B
02EA	CBD9	462	SET 3,C
02EC	CBDA	463	SET 3,D
02EE	CBDB	464	SET 3,E
02F0	CBDC	465	SET 3,H
02F2	CBDD	466	SET 3,L
02F4	CBDE	467	SET 3,(HL)
02F6	CBDF	468	SET 3,A
02F8	CBE0	469	SET 4,B
02FA	CBE1	470	SET 4,C
02FC	CBE2	471	SET 4,D
02FE	CBE3	472	SET 4,E

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0300	CBE4	473	SET 4,H
0302	CBE5	474	SET 4,L
0304	CBE6	475	SET 4,(HL)
0306	CBE7	476	SET 4,A
0308	CBE8	477	SET 5,B
030A	CBE9	478	SET 5,C
030C	CBEA	479	SET 5,D
030E	CBEB	480	SET 5,E
0310	CBEC	481	SET 5,H
0312	CBED	482	SET 5,L
0314	CBEE	483	SET 5,(HL)
0316	CBEF	484	SET 5,A
0318	CBF0	485	SET 6,B
031A	CBF1	486	SET 6,C
031C	CBF2	487	SET 6,D
031E	CBF3	488	SET 6,E
0320	CBF4	489	SET 6,H
0322	CBF5	490	SET 6,L
0324	CBF6	491	SET 6,(HL)
0326	CBF7	492	SET 6,A
0328	CBF8	493	SET 7,B
032A	CBF9	494	SET 7,C
032C	CBFA	495	SET 7,D
032E	CBFB	496	SET 7,E
0330	CBFC	497	SET 7,H
0332	CBFD	498	SET 7,L
0334	CBFE	499	SET 7,(HL)
0336	CBFF	500	SET 7,A
0338	DD09	501	ADD IX,BC
033A	DD19	502	ADD IX,DE
033C	DD218405	503	LD IX,NN
0340	DD228405	504	LD (NN),IX
0344	DD23	505	INC IX
0346	DD29	506	ADD IX,IX
0348	DD2A8405	507	LD IX,(NN)
034C	DD2B	508	DEC IX
034E	DD3405	509	INC (IX + IND)
0351	DD3505	510	DEC (IX + IND)
0354	DD360520	511	LD (IX + IND),N
0358	DD39	512	ADD IX,SP
035A	DD4605	513	LD B,(IX + IND)
035D	DD4E05	514	LD C,(IX + IND)
0360	DD5605	515	LD D,(IX + IND)
0363	DD5E05	516	LD E,(IX + IND)
0366	DD6605	517	LD H,(IX + IND)
0369	DD6E05	518	LD L,(IX + IND)
036C	DD7005	519	LD (IX + IND),B
036F	DD7105	520	LD (IX + IND),C
0372	DD7205	521	LD (IX + IND),D
0375	DD7305	522	LD (IX + IND),E
0378	DD7405	523	LD (IX + IND),H
037B	DD7505	524	LD (IX + IND),L
037E	DD7705	525	LD (IX + IND),A
0381	DD7E05	526	LD A,(IX + IND)
0384	DD8605	527	ADD A,(IX + IND)
0387	DD8E05	528	ADC A,(IX + IND)
038A	DD9605	529	SUB (IX + IND)
038D	DD9E05	530	SBC A,(IX + IND)
0390	DDA605	531	AND (IX + IND)
0393	DDAE05	532	XOR (IX + IND)
0396	DDB605	533	OR (IX + IND)

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0399	DDBE05	534	CP (IX + IND)
039C	DDE1	535	POP IX
039E	DDE3	536	EX (SP),IX
03A0	DDE5	537	PUSH IX
03A2	DDE9	538	JP (IX)
03A4	DDF9	539	LD SP,IX
03A6	DDCB0506	540	RLC (IX + IND)
03AA	DDCB050E	541	RRC (IX + IND)
03AE	DDCB0516	542	RL (IX + IND)
03B2	DDCB051E	543	RR (IX + IND)
03B6	DDCB0526	544	SLA (IX + IND)
03BA	DDCB052E	545	SRA (IX + IND)
03BE	DDCB053E	546	SRL (IX + IND)
03C2	DDCB0546	547	BIT 0,(IX + IND)
03C6	DDCB054E	548	BIT 1,(IX + IND)
03CA	DDCB0556	549	BIT 2,(IX + IND)
03CE	DDCB055E	550	BIT 3,(IX + IND)
03D2	DDCB0566	551	BIT 4,(IX + IND)
03D6	DDCB056E	552	BIT 5,(IX + IND)
03DA	DDCB0576	553	BIT 6,(IX + IND)
03DE	DDCB057E	554	BIT 7,(IX + IND)
03E2	DDCB0586	555	RES 0,(IX + IND)
03E6	DDCB058E	556	RES 1,(IX + IND)
03EA	DDCB0596	557	RES 2,(IX + IND)
03EE	DDCB059E	558	RES 3,(IX + IND)
03F2	DDCB05A6	559	RES 4,(IX + IND)
03F6	DDCB05AE	560	RES 5,(IX + IND)
03FA	DDCB05B6	561	RES 6,(IX + IND)
03FE	DDCB05BE	562	RES 7,(IX + IND)
0402	DDCB05C6	563	SET 0,(IX + IND)
0406	DDCB05CE	564	SET 1,(IX + IND)
040A	DDCB05D6	565	SET 2,(IX + IND)
040E	DDCB05DE	566	SET 3,(IX + IND)
0412	DDCB05E6	567	SET 4,(IX + IND)
0416	DDCB05EE	568	SET 5,(IX + IND)
041A	DDCB05F6	569	SET 6,(IX + IND)
041E	DDCB05FE	570	SET 7,(IX + IND)
0422	ED40	571	IN B,(C)
0424	ED41	572	OUT (C),B
0426	ED42	573	SBC HL,BC
0428	ED438405	574	LD (NN),BC
042C	ED44	575	NEG
042E	ED45	576	RETN
0430	ED46	577	IM 0
0432	ED47	578	LD I,A
0434	ED48	579	IN C,(C)
0436	ED49	580	OUT (C),C
0438	ED4A	581	ADC HL,BC
043A	ED4B8405	582	LD BC,(NN)
043E	ED4D	583	RETI
	ED4F		LD R,A
	ED5F		LD A,R
0440	ED50	584	IN D,(C)
0442	ED51	585	OUT (C),D
0444	ED52	586	SBC HL,DE
0446	ED538405	587	LD (NN),DE
044A	ED56	588	IM I
044C	ED57	589	LD A,I
044E	ED58	590	IN E,(C)
0450	ED59	591	OUT (C),E
0452	ED5A	592	ADC HL,DE
0454	ED5B8405	593	LD DE,(NN)

LOC	OBJ CODE	STMT	SOURCE STATEMENT
045A	ED60	595	IN H,(C)
045C	ED61	596	OUT (C),H
045E	ED62	597	SBC HL,HL
0460	ED67	598	RRD
0462	ED68	599	IN L,(C)
0464	ED69	600	OUT (C),L
0466	ED6A	601	ADC HL,HL
0468	ED6F	602	RLD
046A	ED72	603	SBC HL,SP
046C	ED738405	604	LD (NN),SP
0470	ED78	605	IN A,(C)
0472	ED79	606	OUT (C),A
0474	ED7A	607	ADC HL,SP
0476	ED7B8405	608	LD SP,(NN)
047A	EDA0	609	LDI
047C	EDA1	610	CPI
047E	EDA2	611	INI
0480	EDA3	612	OUTI
0482	EDA8	613	LDD
0484	EDA9	614	CPD
0486	EDAA	615	IND
0488	EDAB	616	OUTD
048A	EDB0	617	LDIR
048C	EDB1	618	CPIR
048E	EDB2	619	INIR
0490	EDB3	620	OTIR
0492	EDB8	621	LDDR
0494	EDB9	622	CPDR
0496	EDBA	623	INDR
0498	EDBB	624	OTDR
049A	FD09	625	ADD IY,BC
049C	FD19	626	ADD IY,DE
049E	FD218405	627	LD IY,NN
04A2	FD228405	628	LD (NN),IY
04A6	FD23	629	INC IY
04A8	FD29	630	ADD IY,IY
04AA	FD2A8405	631	LD IY,(NN)
04AE	FD2B	632	DEC IY
04B0	FD3405	633	INC (IY + IND)
04B3	FD3505	634	DEC (IY + IND)
04B6	FD360520	635	LD (IY + IND),N
04BA	FD39	636	ADD IY,SP
04BC	FD4605	637	LD B,(IY + IND)
04BF	FD3E05	638	LD C,(IY + IND)
04C2	FD5605	639	LD D,(IY + IND)
04C5	FD5E05	640	LD E,(IY + IND)
04C8	FD6605	641	LD H,(IY + IND)
04CB	FD6E05	642	LD L,(IY + IND)
04CE	FD7005	643	LD (IY + IND),B
04D1	FD7105	644	LD (IY + IND),C
04D4	FD7205	645	LD(IY + IND),D
04D7	FD7305	646	LD (IY + IND),E
04DA	FD7405	647	LD (IY + IND),H
04DD	FD7505	648	LD (IY + IND),L
04E0	FD7705	649	LD (IY + IND),A
04E3	FD7E05	650	LD A,(IY + IND)
04E6	FD8605	651	ADD A,(IY + IND)
04E9	FD8E05	652	ADC A,(IY + IND)
04EC	FD9605	653	SUB-(IY + IND)
04EF	FD9E05	654	SBC A,(IY + IND)
04F2	FDA605	655	AND (IY + IND)
04F5	FDAE05	656	XOR (IY + IND)
04F8	FDB605	657	OR (IY + IND)
04FB	FDBE05	658	CP (IY + IND)
04FE	FDE1	659	POP IY
0500	FDE3	660	EX (SP),IY
0502	FDE5	661	PUSH IY
0504	FDE9	662	JP (IY)
0506	FDF9	663	LD SP,IY
0508	FDCB0506	664	RLC (IY + IND)
050C	FDCB050E	665	RRC (IY + IND)
0510	FDCB0516	666	RL (IY + IND)
0514	FDCB051E	667	RR (IY + IND)
0518	FDCB0526	668	SLA (IY + IND)
051C	FDCB052E	669	SRA (IY + IND)
0520	FDCB053E	670	SRL (IY + IND)
0524	FDCB0546	671	BIT 0,(IY + IND)
0528	FDCB054E	672	BIT 1,(IY + IND)
052C	FDCB0556	673	BIT 2,(IY + IND)
0530	FDCB055E	674	BIT 3,(IY + IND)
0534	FDCB0566	675	BIT 4,(IY + IND)
0538	FDCB056E	676	BIT 5,(IY + IND)
053C	FDCB0576	677	BIT 6,(IY + IND)
0540	FDCB057E	678	BIT 7,(IY + IND)
0544	FDCB0586	679	RES 0,(IY + IND)
0548	FDCB058E	680	RES 1,(IY + IND)
054C	FDCB0596	681	RES 2,(IY + IND)
0550	FDCB059E	682	RES 3,(IY + IND)
0554	FDCB05A6	683	RES 4,(IY + IND)
0558	FDCB05AE	684	RES 5,(IY + IND)
055C	FDCB05B6	685	RES 6,(IY + IND)
0560	FDCB05BE	686	RES 7,(IY + IND)
0564	FDCB05C6	687	SET 0,(IY + IND)
0568	FDCB05CE	688	SET 1,(IY + IND)
056C	FDCB05D6	689	SET 2,(IY + IND)
0570	FDCB05DE	690	SET 3,(IY + IND)
0574	FDCB05E6	691	SET 4,(IY + IND)
0578	FDCB05EE	692	SET 5,(IY + IND)
057C	FDCB05F6	693	SET 6,(IY + IND)
0580	FDCB05FE	694	SET 7,(IY + IND)
0584		695 NN	DEFS 2
		696 IND	EQU 5
		697 M	EQU 10H
		698 N	EQU 20H
		699 DIS	EQU 30H
		700	END

Appendix E/Alphabetic List of Instruction Set

Following is an alphabetical listing of the nmemonic or source statement in column four. The object code is shown in column two.

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
0000	8E	1	ADC	A,(HL)
0001	DD8E05	2	ADC	A,(IX + IND)
0004	FD8E05	3	ADC	A,(IY + IND)
0007	8F	4	ADC	A,A
0008	88	5	ADC	A,B
0009	89	6	ADC	A,C
000A	8A	7	ADC	A,D
000B	8B	8	ADC	A,E
000C	8C	9	ADC	A,H
000D	8D	10	ADC	A,L
000E	CE20	11	ADC	A,N
0010	ED4A	12	ADC	HL,BC
0012	ED5A	13	ADC	HL,DE
0014	ED6A	14	ADC	HL,HL
0016	ED7A	15	ADC	HL,SP
0018	86	16	ADD	A,(HL)
0019	DD8605	17	ADD	A,(IX + IND)
001C	FD8605	18	ADD	A,(IY + IND)
001F	87	19	ADD	A,A
0020	80	20	ADD	A,B
0021	81	21	ADD	A,C
0022	82	22	ADD	A,D
0023	83	23	ADD	A,E
0024	84	24	ADD	A,H
0025	85	25	ADD	A,L
0026	C620	26	ADD	A,N
0028	09	27	ADD	HL,BC
0029	19	28	ADD	HL,DE
002A	29	29	ADD	HL,HL
002B	39	30	ADD	HL,SP
002C	DD09	31	ADD	IX,BC
002E	DD19	32	ADD	IX,DE
0030	DD29	33	ADD	IX,IX
0032	DD39	34	ADD	IX,SP
0034	FD09	35	ADD	IY,BC
0036	FD19	36	ADD	IY,DE
0038	FD29	37	ADD	IY,IY
003A	FD39	38	ADD	IY,SP
003C	A6	39	AND	(HL)
003D	DDA605	40	AND	(IX + IND)
0040	FDA605	41	AND	(IY + IND)
0043	A7	42	AND	A
0044	A0	43	AND	B
0045	A1	44	AND	C
0046	A2	45	AND	D
0047	A3	46	AND	E
0048	A4	47	AND	H
0049	A5	48	AND	L
004A	E620	49	AND	N
004C	CB46	50	BIT	0,(HL)
004E	DDCB0546	51	BIT	0,(IX + IND)
0052	FDBC0546	52	BIT	0,(IY + IND)
0056	CB47	53	BIT	0,A
0058	CB40	54	BIT	0,B
005A	CB41	55	BIT	0,C
005C	CB42	56	BIT	0,D
005E	CB43	57	BIT	0,E
0060	CB44	58	BIT	0,H
0062	CB45	59	BIT	0,L
0064	CB4E	60	BIT	1,(HL)
0066	DDCB054E	61	BIT	1,(IX + IND)
006A	FDCB054E	62	BIT	1,(IY + IND)
006E	CB4F	63	BIT	1,A
0070	CB48	64	BIT	1,B
0072	CB49	65	BIT	1,C
0074	CB4A	66	BIT	1.D
0076	CB4B	67	BIT	1,E
0078	CB4C	68	BIT	1,H
007A	CB4D	69	BIT	1,L
007C	CB56	70	BIT	2,(HL)
007E	DDCB0556	71	BIT	2,(IX + IND)
0082	FDCB0556	72	BIT	2,(IY + IND)
0086	CB57	73	BIT	2,A
0088	CB50	74	BIT	2,B
008A	CB51	75	BIT	2,C
008C	CB52	76	BIT	2,D
008E	CB53	77	BIT	2,E
0090	CB54	78	BIT	2,H
0092	CB55	79	BIT	2,L
0094	CB5E	80	BIT	3,(HL)
0096	DDCB055E	81	BIT	3,(IX + IND)
009A	FDCB055E	82	BIT	3,(IY + IND)
009E	CB5F	83	BIT	3,A
00A0	CB58	84	BIT	3,B
00A2	CB59	85	BIT	3,C
00A4	CB5A	86	BIT	3,D
00A6	CB5B	87	BIT	3,E
00A8	CB5C	88	BIT	3,H
00AA	CB5D	89	BIT	3,L
00AC	CB66	90	BIT	4,(HL)
00AE	DDCB0566	91	BIT	4,(IX + IND)
00B2	FDCB0566	92	BIT	4,(IY + IND)
00B6	CB67	93	BIT	4,A
00B8	CB60	94	BIT	4,B
00BA	CB61	95	BIT	4,C
00BC	CB62	96	BIT	4,D
00BE	CB63	97	BIT	4,E
00C0	CB64	98	BIT	4,H
00C2	CB65	99	BIT	4,L
00C4	CB6E	100	BIT	5,(HL)
00C6	DDCB056E	101	BIT	5,(IX + IND)
00CA	FDCB056E	102	BIT	5,(IY + IND)
00CE	CB6F	103	BIT	5,A
00D0	CB68	104	BIT	5,B
00D2	CB69	105	BIT	5,C
00D4	CB6A	106	BIT	5,D
00D6	CB6B	107	BIT	5,E
00D8	CB6C	108	BIT	5,H
00DA	CB6D	109	BIT	5,L
00DC	CB76	110	BIT	6,(HL)

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
00DE	DDCB0576	111	BIT	6,(IX + IND)
00E2	FDCB0576	112	BIT	6,(IY + IND)
00E6	CB77	113	BIT	6,A
00E8	CB70	114	BIT	6,B
00EA	CB71	115	BIT	6,C
00EC	CB72	116	BIT	6,D
00EE	CB73	117	BIT	6,E
00F0	CB74	118	BIT	6,H
00F2	CB75	119	BIT	6,L
00F4	CB7E	120	BIT	7,(HL)
00F6	DDCB057E	121	BIT	7,(IX + IND)
00FA	FDCB057E	122	BIT	7,(IY + IND)
00FE	CB7F	123	BIT	7,A
0100	CB78	124	BIT	7,B
0102	CB79	125	BIT	7,C
0104	CB7A	126	BIT	7,D
0106	CB7B	127	BIT	7,E
0108	CB7C	128	BIT	7,H
010A	CB7D	129	BIT	7,L
010C	DC8405	130	CALL	C,NN
010F	FC8405	131	CALL	M,NN
0112	D48405	132	CALL	NC,NN
0115	CD8405	133	CALL	NN
0118	C48405	134	CALL	NZ,NN
011B	F48405	135	CALL	P,NN
011E	EC8405	136	CALL	PE,NN
0121	E48405	137	CALL	PO,NN
0124	CC8405	138	CALL	Z,NN
0127	3F	139	CCF	
0128	BE	140	CP	(HL)
0129	DDBE05	141	CP	(IX + IND)
012C	FDBE05	142	CP	(IY + IND)
012F	BF	143	CP	A
0130	B8	144	CP	B
0131	B9	145	CP	C
0132	BA	146	CP	D
0133	BB	147	CP	E
0134	BC	148	CP	H
0135	BD	149	CP	L
0136	FE20	150	CP	N
0138	EDA9	151	CPD	
013A	EDB9	152	CPDR	
013C	EDA1	153	CPI	
013E	EDB1	154	CPIR	
0140	2F	155	CPL	
0141	27	156	DAA	
0142	35	157	DEC	(HL)
0143	DD3505	158	DEC	(IX + IND)
0146	FD3505	159	DEC	(IY + IND)
0149	3D	160	DEC	A
014A	05	161	DEC	B
014B	0B	162	DEC	BC
014C	0D	163	DEC	C
014D	15	164	DEC	D
014E	1B	165	DEC	DE
014F	1D	166	DEC	E
0150	25	167	DEC	H
0151	2B	168	DEC	HL
0152	DD2B	169	DEC	IX
0154	FD2B	170	DEC	IY
0156	2D	171	DEC	L

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
0157	3B	172	DEC	SP
0158	F3	173	DI	
0159	102E	174	DJNZ	DIS
015B	FB	175	EI	
015C	E3	176	EX	(SP),HL
015D	DDE3	177	EX	(SP),IX
015F	FDE3	178	EX	(SP),IY
0161	08	179	EX	AF,AF'
0162	EB	180	EX	DE,HL
0163	D9	181	EXX	
0164	76	182	HALT	
0165	ED46	183	IM	0
0167	ED56	184	IM	1
0169	ED5E	185	IM	2
016B	ED78	186	IN	A,(C)
016D	DB20	187	IN	A,N
016F	ED40	188	IN	B,(C)
0171	ED48	189	IN	C,(C)
0173	ED50	190	IN	D,(C)
0175	ED58	191	IN	E,(C)
0177	ED60	192	IN	H,(C)
0179	ED68	193	IN	L,(C)
017B	34	194	INC	(HL)
017C	DD3405	195	INC	(IX + IND)
017F	FD3405	196	INC	(IY + IND)
0182	3C	197	INC	A
0183	04	198	INC	B
0184	03	199	INC	BC
0185	0C	200	INC	C
0186	14	201	INC	D
0187	13	202	INC	DE
0188	1C	203	INC	E
0189	24	204	INC	H
018A	23	205	INC	HL
018B	DD23	206	INC	IX
018D	FD23	207	INC	IY
018F	2C	208	INC	L
0190	33	209	INC	SP
0191	EDAA	210	IND	
0193	EDBA	211	INDR	
0195	EDA2	212	INI	
0197	EDB2	213	INIR	
0199	E9	214	JP	(HL)
019A	DDE9	215	JP	(IX)
019C	FDE9	216	JP	(IY)
019E	DA8405	217	JP	C,NN
01A1	FA8405	218	JP	M,NN
01A4	D28405	219	JP	NC,NN
01A7	C38405	220	JP	NN
01AA	C28405	221	JP	NZ,NN
01AD	F28405	222	JP	P,NN
01B0	EA8405	223	JP	PE,NN
01B3	E28405	224	JP	PO,NN
01B6	CA8405	225	JP	Z,NN
01B9	382E	226	JR	C,DIS
01BB	182E	227	JR	DIS
01BD	302E	228	JR	NC,DIS
01BF	202E	229	JR	NZ,DIS
01C1	282E	230	JR	Z,DIS
01C3	02	231	LD	(BC),A
01C4	12	232	LD	(DE),A

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
01C5	77	233	LD	(HL),A
01C6	70	234	LD	(HL),B
01C7	71	235	LD	(HL),C
01C8	72	236	LD	(HL),D
01C9	73	237	LD	(HL),E
01CA	74	238	LD	(HL),H
01CB	75	239	LD	(HL),L
01CC	3620	240	LD	(HL),N
01CE	DD7705	241	LD	(IX + IND),A
01D1	DD7005	242	LD	(IX + IND),B
01D4	DD7105	243	LD	(IX + IND),C
01D7	DD7205	244	LD	(IX + IND),,D
01DA	DD7305	245	LD	(IX + IND),E
01DD	DD7405	246	LD	(IX + IND),H
01E0	DD7505	247	LD	(IX + IND),L
01E3	DD360520	248	LD	(IX + IND),N
01E7	FD7705	249	LD	(IY + IND),A
01EA	FD7005	250	LD	(IY + IND),B
01ED	FD7105	251	LD	(IY + IND),C
01F0	FD7205	252	LD	(IY + IND),D
01F3	FD7305	253	LD	(IY + IND),E
01F6	FD7405	254	LD	(IY + IND),H
01F9	FD7505	255	LD	(IY + IND),L
01FC	FD360520	256	LD	(IY + IND),N
0200	328405	257	LD	(NN),A
0203	ED438405	258	LD	(NN),BC
0207	ED538405	259	LD	(NN),DE
020B	228405	260	LD	(NN),HL
020E	DD228405	261	LD	(NN),IX
0202	FD228405	262	LD	(NN),IY
0216	ED738405	263	LD	(NN),SP
021A	0A	264	LD	A,(BC)
021B	1A	265	LD	A,(DE)
021C	7E	266	LD	A,(HL)
021D	DD7E05	267	LD	A,(IX + IND)
0220	FD7E05	268	LD	A,(IY + IND)
0223	3A8405	269	LD	A,(NN)
0226	7F	270	LD	A,A
0227	78	271	LD	A,B
0228	79	272	LD	A,C
0229	7A	273	LD	A,D
022A	7B	274	LD	A,E
022B	7C	275	LD	A,H
022C	ED57	276	LD	A,I
022E	7D	277	LD	A,L
022F	3E20	278	LD	A,N
0231	46	279	LD	B,(HL)
0232	DD4605	280	LD	B,(IX + IND)
0235	FD4605	281	LD	B,(IY + IND)
0238	47	282	LD	B,A
0239	40	283	LD	B,B
023A	41	284	LD	B,C
023B	42	285	LD	B,D
023C	43	286	LD	B,E
023D	44	287	LD	B,H
023E	45	288	LD	B,L
023F	0620	289	LD	B,N
0241	ED4B8405	290	LD	BC,(NN)
0245	018405	291	LD	BC,NN
0248	4E	292	LD	C,(HL)
0249	DD4E05	293	LD	C,(IX + IND)
024C	FD4E05	294	LD	C,(IY + IND)
024F	4F	295	LD	C,A
0250	48	296	LD	C,B
0251	49	297	LD	C,C
0252	4A	298	LD	C,D
0253	4B	299	LD	C,E
0254	4C	300	LD	C,H
0255	4D	301	LD	C,L
0256	0E20	302	LD	C,N
0258	56	303	LD	D,(HL)
0259	DD5605	304	LD	D,(IX + IND)
025C	FD5605	305	LD	D,(IY + IND)
025F	57	306	LD	D,A
0260	50	307	LD	D,B
0261	51	308	LD	D,C
0262	52	309	LD	D,D
0263	53	310	LD	D,E
0264	54	311	LD	D,H
0265	55	312	LD	D,L
0266	1620	313	LD	D,N
0268	ED5B8405	314	LD	DE,(NN)
026C	118405	315	LD	DE,NN
026F	5E	316	LD	E,(HL)
0270	DD5E05	317	LD	E,(IX + IND)
0273	FD5E05	318	LD	E,(IY + IND)
0276	5F	319	LD	E,A
0277	58	320	LD	E,B
0278	59	321	LD	E,C
0279	5A	322	LD	E,D
027A	5B	323	LD	E,E
027B	5C	324	LD	E,H
027C	5D	325	LD	E,L
027D	1E20	326	LD	E,N
027F	66	327	LD	H,(HL)
0280	DD6605	328	LD	H,(IX + IND)
0283	FD6605	329	LD	H,(IY + IND)
0286	67	330	LD	H,A
0287	60	331	LD	H,B
0288	61	332	LD	H,C
0289	62	333	LD	H,D
028A	63	334	LD	H,E
028B	64	335	LD	H,H
028C	65	336	LD	H,L
028D	2620	337	LD	H,N
028F	2A8405	338	LD	HL,(NN)
0292	218405	339	LD	HL,NN
0295	ED47	340	LD	I,A
0297	DD2A8405	341	LD	IX,(NN)
029B	DD218405	342	LD	IX,NN
029F	FD2A8405	343	LD	IY,(NN)
02A3	FD218405	344	LD	IY,NN
02A7	6E	345	LD	L,(HL)
02A8	DD6E05	346	LD	L,(IX + IND)
02AB	FD6E05	347	LD	L,(IY + IND)
02AE	6F	348	LD	L,A
02AF	68	349	LD	L,B
02B0	69	350	LD	L,C
02B1	6A	351	LD	L,D
02B2	6B	352	LD	L,E
02B3	6C	353	LD	L,H
02B4	6D	354	LD	L,L

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
02B4	6D	354	LD	L,L
02B5	2E20	355	LD	L,N
	ED4F		LD	R,A
02B7	ED7B8405	356	LD	SP,(NN)
02BB	F9	357	LD	SP,HL
02BC	DDF9	358	LD	SP,IX
02BE	FDF9	359	LD	SP,IY
02C0	318405	360	LD	SP,NN
02C3	EDA8	361	LDD	
02C5	EDB8	362	LDDR	
02C7	EDA0	363	LDI	
02C9	EDB0	364	LDIR	
02CB	ED44	365	NEG	
02CD	00	366	NOP	
02CE	B6	367	OR	(HL)
02CF	DDB605	368	OR	(IX + IND)
02D2	FDB605	369	OR	(IY + IND)
02D5	B7	370	OR	A
02D6	B0	371	OR	B
02D7	B1	372	OR	C
02D8	B2	373	OR	D
02D9	B3	374	OR	E
02DA	B4	375	OR	H
02DB	B5	376	OR	L
02DC	F620	377	OR	N
02DE	ED8B	378	OTDR	
02E0	EDB3	379	OTIR	
02E2	ED79	380	OUT	(C),A
02E4	ED41	381	OUT	(C),B
02E6	ED49	382	OUT	(C),C
02E8	ED51	383	OUT	(C),D
02EA	ED59	384	OUT	(C),E
02EC	ED61	385	OUT	(C),H
02EE	ED69	386	OUT	(C),L
02F0	D320	387	OUT	N,A
02F2	EDAB	388	OUTD	
02F4	EDA3	389	OUTI	
02F6	F1	390	POP	AF
02F7	C1	391	POP	BC
02F8	D1	392	POP	DE
02F9	E1	393	POP	HL
02FA	DDE1	394	POP	IX
02FC	FDE1	395	POP	IY
02FE	F5	396	PUSH	AF
02FF	C5	397	PUSH	BC
0300	D5	398	PUSH	DE
0301	E5	399	PUSH	HL
0302	DDE5	400	PUSH	IX
0304	FDE5	401	PUSH	IY
0306	CB86	402	RES	0,(HL)
0308	DDCB0586	403	RES	0,(IX + IND)
030C	FDCB0586	404	RES	0,(IY + IND)
0310	CB87	405	RES	0,A
0312	CB80	406	RES	0,B
0314	CB81	407	RES	0,C
0316	CB82	408	RES	0,D
0318	CB83	409	RES	0,E
031A	CB84	410	RES	0,H
031C	CB85	411	RES	0,L
031E	CB8E	412	RES	1,(HL)
0320	DDCB058E	413	RES	1,(IX + IND)

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
0324	FDCB058E	414	RES	1,(IY + IND)
0328	CB8F	415	RES	1,A
032A	CB88	416	RES	1,B
032C	CB89	417	RES	1,C
032E	CB8A	418	RES	1,D
0330	CB8B	419	RES	1,E
0332	CB8C	420	RES	1,H
0334	CB8D	421	RES	1,L
0336	CB96	422	RES	2,(HL)
0338	DDCB0596	423	RES	2,(IX + IND)
033C	FDCB0596	424	RES	2,(IY + IND)
0340	CB97	425	RES	2,A
0342	CB90	426	RES	2,B
0344	CB91	427	RES	2,C
0346	CB92	428	RES	2,D
0348	CB93	429	RES	2,E
034A	CB94	430	RES	2,H
034C	CB95	431	RES	2,L
034E	CB9E	432	RES	3,(HL)
0350	DDCB059E	433	RES	3,(IX + IND)
0354	FDCB059E	434	RES	3,(IY + IND)
0358	CB9F	435	RES	3,A
035A	CB98	436	RES	3,B
035C	CB99	437	RES	3,C
035E	CB9A	438	RES	3,D
0360	CB9B	439	RES	3,E
0362	CB9C	440	RES	3,H
0364	CB9D	441	RES	3,L
0366	CBA6	442	RES	4,(HL)
0368	DDCB05A6	443	RES	4,(IX + IND)
036C	FDCB05A6	444	RES	4,(IY + IND)
0370	CBA7	445	RES	4,A
0372	CBA0	446	RES	4,B
0374	CBA1	447	RES	4,C
0376	CBA2	448	RES	4,D
0378	CBA3	449	RES	4,E
037A	CBA4	450	RES	4,H
037C	CBA5	451	RES	4,L
037E	CBAE	452	RES	5,(HL)
0380	DDCB05AE	453	RES	5,(IX + IND)
0384	FDCB05AE	454	RES	5,(IY + IND)
0388	CBAF	455	RES	5,A
038A	CBA8	456	RES	5,B
038C	CBA9	457	RES	5,C
038E	CBAA	458	RES	5,D
0390	CBAB	459	RES	5,E
0392	CBAC	460	RES	5,H
0394	CBAD	461	RES	5,L
0396	CBB6	462	RES	6,(HL)
0398	DDCB05B6	463	RES	6,(IX + IND)
039C	FDCB05B6	464	RES	6,(IY + IND)
03A0	CBB7	465	RES	6,A
03A2	CBB0	466	RES	6,B
03A4	CBB1	467	RES	6,C
03A6	CBB2	468	RES	6,D
03A8	CBB3	469	RES	6,E
03AA	CBB4	470	RES	6,H
03AC	CBB5	471	RES	6,L
03AE	CBBE	472	RES	7,(HL)
03B0	DDCB05BE	473	RES	7,(IX + IND)
03B4	FDCB05BE	474	RES	7,(IY + IND)

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
03B8	CBBF	475	RES	7,A
03BA	CBB8	476	RES	7,B
03BC	CBB9	477	RES	7,C
03BE	CBBA	478	RES	7,D
03C0	CBBB	479	RES	7,E
03C2	CBBC	480	RES	7,H
03C4	CBBD	481	RES	7,L
03C6	C9	482	RET	
03C7	D8	483	RET	C
03C8	F8	484	RET	M
03C9	D0	485	RET	NC
03CA	C0	486	RET	NZ
03CB	F0	487	RET	P
03CC	E8	488	RET	PE
03CD	E0	489	RET	PO
03CE	C8	490	RET	Z
03CF	ED4D	491	RETI	
03D1	ED45	492	RETN	
03D3	CB16	493	RL	(HL)
03D5	DDCB0516	494	RL	(IX + IND)
03D9	FDCB0516	495	RL	(IY + IND)
03DD	CB17	496	RL	A
03DF	CB10	497	RL	B
03E1	CB11	498	RL	C
03E3	CB12	499	RL	D
03E5	C813	500	RL	E
03E7	CB14	501	RL	H
03E9	CB15	502	RL	L
03EB	17	503	RLA	
03EC	CB06	504	RLC	(HL)
03EE	DDCB0506	505	RLC	(IX + IND)
03F2	FDCB0506	506	RLC	(IY + IND)
03F6	CB07	507	RLC	A
03F8	CB00	508	RLC	B
03FA	CB01	509	RLC	C
03FC	CB02	510	RLC	D
03FE	CB03	511	RLC	E
0400	CB04	512	RLC	H
0402	CB05	513	RLC	L
0404	07	514	RLCA	
0405	ED6F	515	RLD	
0407	CB1E	516	RR	(HL)
0409	DDCB051E	517	RR	(IY + IND)
040D	FDCB051E	518	RR	(IY + IND)
0411	CB1F	519	RR	A
0413	CB18	520	RR	B
0415	CB19	521	RR	C
0417	CB1A	522	RR	D
0419	CB1B	523	RR	E
041B	CB1C	524	RR	H
041D	CB1D	525	RR	L
041F	1F	526	RRA	
0420	CB0E	527	RRC	(HL)
0422	DDCB050E	528	RRC	(IX + IND)
0426	FDCB050E	529	RRC	(IY + IND)
042A	CB0F	530	RRC	A
042C	CB08	531	RRC	B
042E	CB09	532	RRC	C
0430	CB0A	533	RRC	D
0432	CB0B	534	RRC	E
0434	CB0C	535	RRC	H
0436	CB0D	536	RRC	L
0438	0F	537	RRCA	
0439	ED67	538	RRD	
043B	C7	539	RST	0
043C	D7	540	RST	10H
043D	DF	541	RST	18H
043E	E7	542	RST	20H
043F	EF	543	RST	28H
0440	F7	544	RST	30H
0441	FF	545	RST	38H
0442	CF	546	RST	08H
0443	9E	547	SBC	A,(HL)
0444	DD9E05	548	SBC	A,(IX + IND)
0447	FD9E05	549	SBC	A,(IY + IND)
044A	9F	550	SBC	A,A
044B	98	551	SBC	A,B
044C	99	552	SBC	A,C
044D	9A	553	SBC	A,D
044E	9B	554	SBC	A,E
044F	9C	555	SBC	A,H
0450	9D	556	SBC	A,L
0451	DE20	557	SBC	A,N
0453	ED42	558	SBC	HL,BC
0455	ED52	559	SBC	HL,DE
0457	ED62	560	SBC	HL,HL
0459	ED72	561	SBC	HL,SP
045B	37	562	SCF	
045C	CBC6	563	SET	0,(HL)
045E	DDCB05C6	564	SET	0,(IX + IND)
0462	FDCB05C6	565	SET	0,(IY + IND)
0466	CBC7	566	SET	0,A
0468	CBC0	567	SET	0,B
046A	CBC1	568	SET	0,C
046C	CBC2	569	SET	0,D
046E	CBC3	570	SET	0,E
0470	CBC4	571	SET	0,H
0472	CBC5	572	SET	0,L
0474	CBCE	573	SET	1,(HL)
0476	DDCB05CE	574	SET	1,(IX + IND)
047A	FDCB05CE	575	SET	1,(IY + IND)
047E	CBCF	576	SET	1,A
0480	CBC8	577	SET	1,B
0482	CBC9	578	SET	1,C
0484	CBCA	579	SET	1,D
0486	CBCB	580	SET	1,E
0488	CBCC	581	SET	1,H
048A	CBCD	582	SET	1,L
048C	CBD6	583	SET	2,(HL)
048E	DDCB05D6	584	SET	2,(IX + IND)
0492	FDCB05D6	585	SET	2,(IY + IND)
0496	CBD7	586	SET	2,A
0498	CBD0	587	SET	2,B
049A	CBD1	588	SET	2,C
049C	CBD2	589	SET	2,D
049E	CBD3	590	SET	2,E
04A0	CBD4	591	SET	2,H
04A2	CBD5	592	SET	2,L
04A4	CBD8	593	SET	3,B
04A6	CBDE	594	SET	3,(HL)
04A8	DDCB05DE	595	SET	3,(IX + IND)
04AC	FDCB05DE	596	SET	3,(IY + IND)

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
04B4	CBDA	599	SET	3,D
04B6	CBDB	600	SET	3,E
04B8	CBDC	601	SET	3,H
04BA	CBDD	602	SET	3,L
04BC	CBE6	603	SET	4,(HL)
04BE	DDCB05E6	604	SET	4,(IX + IND)
04C2	FDCB05E6	605	SET	4,(IY + IND)
04C6	CBE7	606	SET	4,A
04C8	CBE0	607	SET	4,B
04CA	CBE1	608	SET	4,C
04CC	CBE2	609	SET	4,D
04CE	CBE3	610	SET	4,E
04D0	CBE4	611	SET	4,H
04D2	CBE5	612	SET	4,L
04D4	CBEE	613	SET	5,(HL)
04D6	DDCB05EE	614	SET	5,(IX + IND)
04DA	FDCB05EE	615	SET	5,(IY + IND)
04DE	CBEF	616	SET	5,A
04E0	CBE8	617	SET	5,B
04E2	CBE9	618	SET	5,C
04E4	CBEA	619	SET	5,D
04E6	CBEB	620	SET	5,E
04E8	CBEC	621	SET	5,H
04EA	CBED	622	SET	5,L
04EC	CBF6	623	SET	6,(HL)
04EE	DDCB05F6	624	SET	6,(IX + IND)
04F2	FDCB05F6	625	SET	6,(IY + IND)
04F6	CBF7	626	SET	6,A
04F8	CBF0	627	SET	6,B
04FA	CBF1	628	SET	6,C
04FC	CBF2	629	SET	6,D
04FE	CBF3	630	SET	6,E
0500	CBF4	631	SET	6,H
0502	CBF5	632	SET	6,L
0504	CBFE	633	SET	7,(HL)
0506	DDCB05FE	634	SET	7,(IX + IND)
050A	FDCB05FE	635	SET	7,(IY + IND)
050E	CBFF	636	SET	7,A
0510	CBF8	637	SET	7,B
0512	CF9	638	SET	7,C
0514	CBFA	639	SET	7,D
0516	CBFB	640	SET	7,E
0518	CBFC	641	SET	7,H
051A	CBFD	642	SET	7,L
051C	CB26	643	SLA	(HL)
051E	DDCB0526	644	SLA	(IX + IND)
0522	FDCB0526	645	SLA	(IY + IND)
0526	CB27	646	SLA	A
0528	CB20	647	SLA	B
052A	CB21	648	SLA	C
052C	CB22	649	SLA	D

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
052E	CB23	650	SLA	E
0530	CB24	651	SLA	H
0532	CB25	652	SLA	L
0534	CB2E	653	SRA	(HL)
0536	DDCB052E	654	SRA	(IX + IND)
053A	FDCB052E	655	SRA	(IY + IND)
053E	CB2F	656	SRA	A
0540	CB28	657	SRA	B
0542	CB29	658	SRA	C
0544	CB2A	659	SRA	D
0546	CB2B	660	SRA	E
0548	CB2C	661	SRA	H
054A	CB2D	662	SRA	L
054C	CB3E	663	SRL	(HL)
054E	DDCB053E	664	SRL	(IX + IND)
0552	FDCB053E	665	SRL	(IY + IND)
0556	CB3F	666	SRL	A
0558	CB38	667	SRL	B
055A	CB39	668	SRL	C
055C	CB3A	669	SRL	D
055E	CB3B	670	SRL	E
0560	CB3C	671	SRL	H
0562	CB3D	672	SRL	L
0564	96	673	SUB	(HL)
0565	DD9605	674	SUB	(IX + IND)
0568	FD9605	675	SUB	(IY + IND)
056B	97	676	SUB	A
056C	90	677	SUB	B
056D	91	678	SUB	C
056E	92	679	SUB	D
056F	93	680	SUB	E
0570	94	681	SUB	H
0571	95	682	SUB	L
0572	D620	683	SUB	N
0574	AE	684	XOR	(HL)
0575	DDAE05	685	XOR	(IX + IND)
0578	FDAE05	686	XOR	(IY + IND)
057B	AF	687	XOR	A
057C	A8	688	XOR	B
057D	A9	689	XOR	C
057E	AA	690	XOR	D
057F	AB	691	XOR	E
0580	AC	692	XOR	H
0581	AD	693	XOR	L
0582	EE20	694	XOR	N
0584		695 NN	DEFS	2
		696 IND	EQU	5
		697 M	EQU	10H
		698 N	EQU	20H
		699 DIS	EQU	30H
		700	END	

AppendixF / Z-80 CPU Register and Architecture

This section gives information about the actual Z80 chip including the Central Processing Unit (CPU) Register configuration.

Z-80 CPU Architecture

A block diagram of the internal architecture of the Z-80 CPU is shown in **Figure 2**. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.

CPU Registers

The Z-80 CPU contains 208 bits of R/W memory that are accessible to the programmer. **Figure 3** illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z-80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs of 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

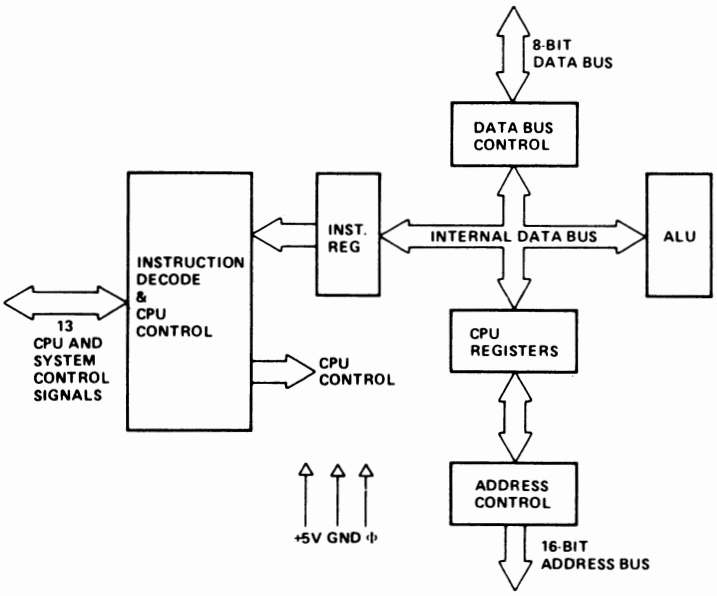


Figure 2, Z-80 CPU Block Diagram.

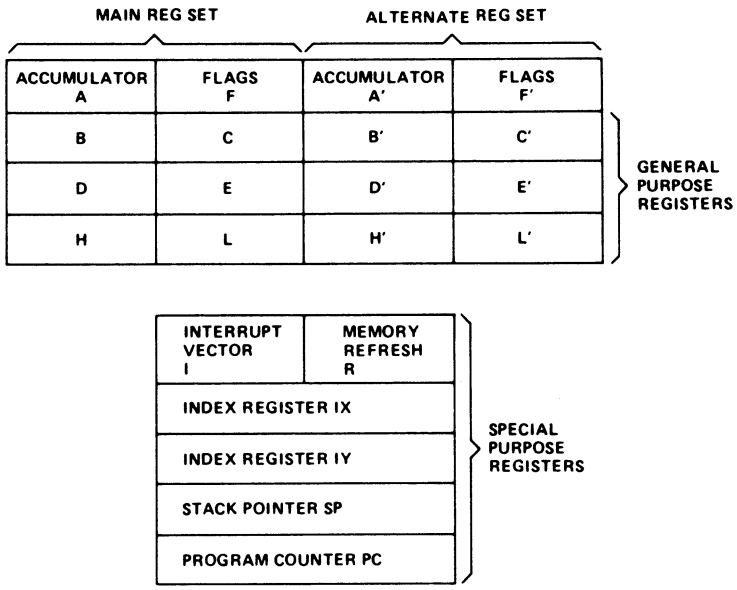


Figure 3, Z-80 CPU Register Configuration.

1. **Program Counter (PC).** The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.

2. **Stack Pointer (SP).** The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file.

   Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

3. **Two Index Register (IX & IY).** The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

4. **Interrupt Page Address Register (I).** The Z-80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.
5. **Memory Refresh Register (R).** The Z-80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. Seven bits of this 8 bit register are automatically incremented after each instruction fetch. The eighth bit will remain as programmed as the result of an LD R, A instruction. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer. During refresh, the contents of the I register are placed on the upper 8 bits of the address bus.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with a single exchange instruction so that he may easily work with either pair.

General Purpose Registers

There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs by the programmer. One set is called BC, DE and HL while the complementary set is called BC,' DE and HL.' At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify programming, especially in ROM based systems where little external read/write memory is available.

Arithmetic & Logic Unit (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

Add	Left or right shifts or rotates (arithmetic and logical)
Subtract	Increment
Logical AND	Decrement
Logical OR	Set bit
Logical Exclusive OR	Reset bit
Compare	Test Bit

Instruction Register and CPU Control

As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control sections performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, control the ALU and provide all required external control signals.

Z-80 CPU Pin Description

The Z-80 CPU is packaged in an industry standard 40 pin Dual In-Line Package. The I/O pins are shown in **Figure 4** and the function of each is described below.

A_0-A_{15} (Address Bus)	Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the 8 lower address bits to allow the user to directly select up to 256 input or 256 output ports. A_0 is the least significant address bit. During refresh time, the lower 7 bits contain a valid refresh address.
D_0-D_7 (Data Bus)	Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.
$\overline{M_1}$ (Machine Cycle one)	Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution. Note that during execution of 2-byte op-codes, $\overline{M_1}$ is generated as each op-code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH or FDH. $\overline{M_1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

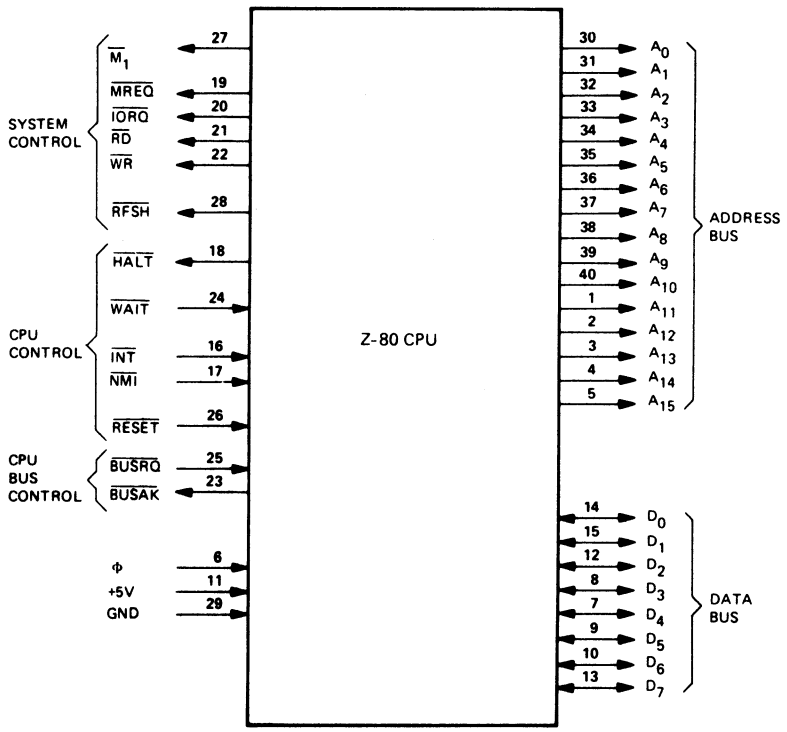


Figure 4, Z-80 Pin Configuration.

$\overline{\text{MREQ}}$ (Memory Request)	Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.
$\overline{\text{IORQ}}$ (Input/Output Request)	Tri-state output, active low. The $\overline{\text{IORQ}}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{\text{IORQ}}$ signal is also generated with an $\overline{\text{M}_1}$ signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.
$\overline{\text{RD}}$ (Memory Read)	Tri-state output, active low. $\overline{\text{RD}}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.
$\overline{\text{WR}}$ (Memory Write)	Tri-state output, active low. $\overline{\text{WR}}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{RFSH}$ (Refresh)	Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories.
$\overline{HALT}$ (Halt state)	Output, active low. $\overline{HALT}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.
$\overline{WAIT}$ (Wait)	Input, active low. $\overline{WAIT}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.
$\overline{INT}$ (Interrupt Request)	Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{BUSRQ}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{IORQ}$ during M_1 time) is sent out at the beginning of the next instruction cycle.
$\overline{NMI}$ (Non Maskable Interrupt)	Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z-80 CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that a $\overline{BUSRQ}$ will override a $\overline{NMI}$.
$\overline{RESET}$	Input, active low. $\overline{RESET}$ forces the program counter to zero and initializes the CPU. The CPU initialization includes: 1) Disable the interrupt enable flip-flop 2) Set Register I = 00_H 3) Set Register R = 00_H 4) Set Interrupt Mode 0 During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state.

$\overline{\text{BUSRQ}}$ (Bus Request)	Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{\text{BUSRQ}}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.
$\overline{\text{BUSAK}}$ (Bus Acknowledge)	Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.
Φ	Single phase TTL level clock which requires only a 330 ohm pull-up resistor to +5 volts to meet all clock requirements.

Z-80 CPU Instruction Set

The Z-80 CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions can be broken down into the following major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (set, reset, test)
- Jump, Call and Return
- Input/Output
- Basic CPU Control

INDEX

Subject	Page
Abbreviations	17
Accumulator	248
ADC A,S	109
ADC HL,ss	142
Add/Subtract flag	231
ADD A,(HL)	107
ADD A,(IX+d)	107
ADD A,n	106
ADD A,r	105
ADD HL,ss	141
ADD IX,pp	144
ADD A,(IY+d)	108
ADD IY,rr	145
Alphabetical list of Z-80 instructions	240-245
AND s	115
Arithmetic logic unit (ALU)	249
Assembler	21
Commands	21
Definitions	25-26
Output	23
Switch	21-22
Using the assembler	21
BIT B,(HL)	178
BIT B,(IX+d)	179
BIT B,(IY+d)	180
BIT b,r	177
CALL cc,nn	202
CALL nn	201
Carry flag	231
CCF	134
Central processing unit (CPU)	249
Comments	26
Computer Type	4
CPD	102
CPDR	103
CPI	99
CPIR	101
CPL	132
CP s	122
CPU block diagram	246
CPU—pin description	249-252
Current line	7

Subject	Page
DAA	131
DEC IX	149
DEC IY	149
DEC m	127
DEC ss	148
DI	136
DJNZ e	199
Editor	
Commands	8-15, 18, 19
Definition	1
Features of	2
How to use	2, 5, 7
EI	137
Error messages (assembler)	24-25
Error messages (Editor)	16
EX AF,AF'	87
EX DE,HL	87
Expressions	29
EX (SP),HL	89
EX (SP),IX	90
EX (SP),IY	91
EXX	88
File	7
File name	7
Flag register	248
Flags (status)	231
Half-Carry flag	232-233
HALT	136
IM 0	138
IM 1	138
IM 2	139
IN A,(n)	211
INC (HL)	125
INC IX	146
INC (IX+d)	125
INC IY	147
INC (IY+d)	126
INC r	124
Increment	7
INC ss	146
IND	216
Index registers	247
INDR	217

Subject	Page
INI	213
INIR	214
Input/Output commands	13
IN r,(C)	212
Interrupt register	248
Italic type	4
JP cc,nn	190
JP (HL)	197
JP (IX)	198
JP (IY)	198
JP nn	189
JR C,e	192
JR e	191
JR NC,e	193
JR NZ,e	195
JR Z,e	194
Label	26
LD A,(BC)	57
LD A,(DE)	57
LD A,I	61
LD A,(nn)	58
LD A,R	62
LD (BC),A	59
LDD	96
LD dd,(nn)	68
LD dd,nn	65
LD (DE),A	59
LDDR	97
LD (HL),n	54
LD HL,(nn)	67
LD (HL),r	52
LDI	93
LD I,A	62
LDIR	94
LD (IX+d),n	55
LD (IX+d),r	52
LD IX,nn	66
LD IX,(nn)	69
LD (IY+d),n	56
LD (IY+d),r	53
LD IY,nn	67
LD IY,(nn)	70
LD (nn),A	60
LD (nn),dd	72
LD (nn),HL	71

Subject	Page
LD (nn),IX	73
LD (nn),IY	74
LD R,A	63
LD r,(HL)	49
LD r,(IX+d)	49
LD r,(IY+d)	51
LD r,n	48
LD r,r'	47
LD SP,HL	75
LD SP,IX	76
LD SP,IY	77
Memory refresh register	248
Mnemonics	26
Model I—Subroutines	228
NEG	133
NOP	135
Notations	4
Numerical list of Z-80 instructions	234-239
Object code	33
Object code	4
Object file	4
Operands	26, 29
Operations	27
OR s	117
OTDR	224
OTIR	221
OUT (C),r	219
OUTD	223
OUTI	220
OUT (n),A	218
Parity/Overflow flag	232
POP IX	82
POP IY	84
POP qq	81
Program counter	247
Pseudo Operations	27-28
PUSH IX	79
PUSH IY	80
PUSH qq	77
Register configuration	246
RES b,m	186
RET	204
RET cc	205
RETI	207
RETN	208

Subject	Page

RLA 152
RLCA 151
RLC (HL) 156
RLC (IX+d) 157
RLC (IY+d) 158
RLC r 155
RLD 173
RL m 160
RRA 154
RRCA 153
RRC m 162
RRD 175
RR m 164
RST p 209
Sample Programming 31-36
SBC A,s 113
SBC HL,ss 143
SCF 135
SET b,(HL) 182
SET b,(IX+d) 183
SET b,(IY+d) 185
SET b,r 181
Sign flag 233
Using the TPSRC utility 227
SLA m 166
Source Code 4
Source File 4
Special keys 8, 17
Special Terms 4
SRA m 169
SRL m 171
Stack Pointer 247
Status flags 231
SUB s 111
Symbols 17
Text 7
Text buffer 7
Text handling 7
Text handling commands 9
Trial Assembly 32
XOR s 119
Z-80 instructions 37-226
Zero flag 233

IMPORTANT NOTICE

ALL RADIO SHACK COMPUTER PROGRAMS ARE LICENSED ON AN "AS IS" BASIS WITHOUT WARRANTY.

Radio Shack shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by Radio Shack, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

NOTE: Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a period of time adequate to insure that results of operation of the computer or program are satisfactory.

RADIO SHACK SOFTWARE LICENSE

A. Radio Shack grants to CUSTOMER a non-exclusive, paid up license to use on CUSTOMER'S computer the Radio Shack computer software received. Title to the media on which the software is recorded (cassette and/or disk) or stored (ROM) is transferred to the CUSTOMER, but not title to the software.

B. In consideration for this license, CUSTOMER shall not reproduce copies of Radio Shack software except to reproduce the number of copies required for use on CUSTOMER'S computer (if the software allows a back-up copy to be made), and shall include Radio Shack's copyright notice on all copies of software reproduced in whole or in part.

C. CUSTOMER may resell Radio Shack's system and applications software (modified or not, in whole or in part), provided CUSTOMER has purchased one copy of the software for each one resold. The provisions of this software License (paragraphs A, B, and C) shall also be applicable to third parties purchasing such software from CUSTOMER.

RADIO SHACK TC **A DIVISION OF TANDY CORPORATION**

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U. K.
280-316 VICTORIA ROAD RYDALMERE, N.S.W. 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

Printed in U.S.A.